VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

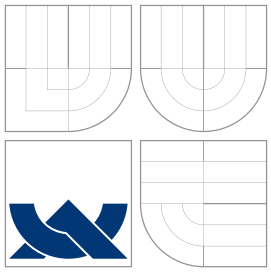
GOOGLE GADGETS V PROSTŘEDÍ EXT JS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

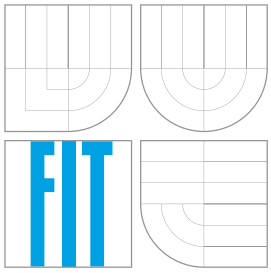
AUTOR PRÁCE
AUTHOR

PETR DVOŘÁK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GOOGLE GADGETS V PROSTŘEDÍ EXT JS

GOOGLE GADGETS IN EXT JS ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR DVOŘÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV DYTRYCH
SUPERVISOR

BRNO 2010

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2009/2010

Zadání bakalářské práce

Řešitel: **Dvořák Petr**
Obor: Informační technologie
Téma: **Google Gadgets v prostředí Ext JS**
Google Gadgets in Ext JS Environment
Kategorie: Web

Pokyny:

1. Seznamte se s možnostmi nástroje Ext JS, API Google Gadgets a správcem rozložení stránky projektu KiWi.
2. Prozkoumejte omezení kompatibility správce rozložení stránky projektu KiWi s Google Gadgets, přičemž se zaměřte na problematiku určování identifikace komponentu uživatelského rozhraní.
3. Navrhněte rozšíření správce rozložení pro dosažení maximální možné kompatibility s Google Gadgets tak, aby nebyla nutná úprava skriptů v Gadgets.
4. Implementujte navržené řešení a srovnejte rychlost načtení a sestavení stránky před a po úpravě správce rozložení.
5. Zhodnoťte dosažené výsledky a porovnejte zvolený přístup s alternativními metodami.

Literatura:

- podle dohody

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dytrych Jaroslav, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2009
Datum odevzdání: 19. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2
L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce analyzuje platformu Google Gadgets a existující řešení správce rozložení stránky projektu KiWi. Stručně obě platformy popisuje a zaměřuje se na návrh a implementaci začlenění podpory Google Gadgets do správce rozložení. Na závěr srovnává původní a upravené řešení z hlediska rychlosti načítání stránky.

Abstract

This thesis aims to analyse the Google Gadgets platform and an existing system of a page layout manager of the KiWi project. It provides a brief description of both platforms and focuses on a design and an implementation of the Google Gadgets support inside the page layout manager. Finally it compares the original with the modified solution in terms of page loading speed.

Klíčová slova

Google Gadgets, uživatelské rozhraní, web, HTML, JavaScript, ExtJS, komponenty, rozložení webové stránky

Keywords

Google Gadgets, user interface, web, HTML, JavaScript, ExtJS, components, web page layout

Citace

Petr Dvořák: Google Gadgets v prostředí Ext JS, bakalářská práce, Brno, FIT VUT v Brně, 2010

Google Gadgets v prostředí Ext JS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Dytrycha.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Dvořák
16. května 2010

Poděkování

Je příjemnou povinností poděkovat na tomto místě vedoucímu mé práce, Ing. Jaroslavu Dytrychovi, za odborné vedení a cenné rady. Poděkování patří také kolegům vývojářům, kteří se o své znalosti podílejí, ať už v odborné literatuře, nebo formou článků na Internetu. A v neposlední řadě mé rodině, která mne podporuje po celý dosavadní život. Děkuji.

Obsah

KAPITOLA 1

Úvod

Cílem této práce je implementovat podporu pro Google Gadgets do existujícího systému správce rozložení stránky projektu KiWi.

Požadavky od uživatelů a vývojového týmu systému, které musí splňovat výsledná verze, zmiňuje kapitola 2 – Analýza a specifikace požadavků.

Aby bylo možné podporu pro Google Gadgets začlenit, je nejdříve nutné obě platformy analyzovat. Stručnému popisu správce rozložení stránky, jeho vlastnostem, které jsou pro rozšíření funkcionality podstatné, a současnému řešení se věnuje kapitola 3 – Analýza správce rozložení stránky projektu KiWi. Analýze systému Google Gadgets se pak věnuje kapitola 4 – Analýza Google Gadgets, která je rozdělena na analýzu platformy jako takové a na analýzu javascriptového API, které platforma používá.

Na základě provedené analýzy byly zvoleny technologie, které se při vývoji použijí. Jejich výčet a stručnou charakteristiku lze nalézt v kapitole 5 – Použité technologie a standardy. Vlastní návrh řešení (kapitola 6) zmiňuje i alternativní přístupy včetně jejich výhod a nevýhod. Zaměřuje se na problém identifikace komponent, na způsob začlenění gadgetů do správce rozložení a na implementaci javascriptového API.

Popis implementace navrženého řešení je uveden v kapitole 7 – Implementace. Zhodnocení výkonnosti a testování systému je zmíněno v poslední kapitole, kapitole 8 – Testování.

Součástí jsou i dvě přílohy, které aktualizují původní dokumentaci ke správci rozložení stránky. Jedná se o přílohu A – Dokumentace pro vývojáře vytvářející komponenty, která popisuje vývoj komponent, a o přílohu B – Požadavky na server, která zmiňuje požadavky na produkční server se správcem rozložení.

KAPITOLA 2

Analýza a specifikace požadavků

Cílem tohoto projektu je navrhnout a implementovat podporu pro Google Gadgets do existujícího správce rozložení stránky projektu KiWi. Tato implementace musí odpovídat specifikaci Gadgets API tak, aby byla zajištěna maximální možná kompatibilita s již vytvořenými Google Gadgets. Implementované řešení je následně nutné porovnat s původní verzí správce rozložení stránky, a to zejména v rychlosti načítání a sestavení stránky.

Dochází tedy k rozšiřování funkčnosti již existujícího systému. Je nutné brát ohled na zpětnou kompatibilitu a v některých ohledech vycházet z předem jasně stanovených pravidel, která nelze změnit. To se týká zejména rozšíření správce rozložení stránky, kdy bude nutné provést rozšíření programového kódu – tj. použít stejnou technologii, kterou je správce rozložení stránky vytvořen – a zachovat současné API. Další omezení jsou kladena na podporu programového vybavení uživatelů, kteří správce rozložení stránky používají, a v neposlední řadě je také nutné respektovat požadavky, které přicházejí od správců systému.

Volba technologie a vlastní způsob začlenění Google Gadgets nejsou dané a úkolem této práce je navrhnout možná řešení. Protože jsou ovšem správce rozložení stránky i Google Gadgets webové platformy, je od začátku zřejmé, že i toto rozšíření bude postavené na webových technologiích.

2.1 Výhradně klientská aplikace

Klíčovým požadavkem je, aby téměř veškerá podpora běžela na straně klienta – řešení nesmí mít žádnou rozsáhlou serverovou část. A to zejména z důvodu, že serverovou část projektu není možné nijak ovlivňovat. Jednoduché podpůrné skripty jsou povoleny, protože ty lze v případě, kdy není potřebná technologie na serveru k dispozici, poměrně jednoduchým způsobem pro podporovanou technologii přepracovat.

2.2 Požadovaná programová podpora

Dalším klíčovým požadavkem je prostředí, ve kterém má systém fungovat. V současné verzi je správce rozložení stránky dle dokumentace [27] podporován ve webových prohlížečích Mozilla Firefox 2 a 3, Microsoft Internet Explorer 6 a 7, Opera 9.21 a Safari 3.1.2. Tuto podporu je u prohlížečů Mozilla Firefox a Internet Explorer nutné zachovat a rozšířit i na nejnovější verze, u ostatních prohlížečů (Opera, Safari a nově i Google Chrome) stačí podporovat nejnovější dostupnou verzi daného prohlížeče.

Na podporu externích zásuvných modulů (jakými jsou například Adobe Flash Player, Java Runtime Environment nebo Microsoft Silverlight) se nelze spoléhat a jejich použití jako primárních technologií není doporučeno.

Vzhledem k povaze projektu se na straně uživatele vždy očekává zapnutá podpora JavaScriptu, aktivní kaskádové styly (CSS) i zobrazené obrázky. Uživatelé systému tvoří uzavřenou a předem proškolenou komunitu, a není proto nutné začleňovat žádné nové speciální prvky přístupnosti.

2.3 Plná kontrola nad obsahem

Při návrhu a vývoji lze vycházet z toho, že správce serveru (nebo uživatel systému) má plnou kontrolu nad obsahem, který zobrazuje. Tím je míněno, že veškerý obsah, který bude správce rozložení stránky zpracovávat, bude předem prověřený, schválený a případně jednoduchým způsobem ručně upravený tak, aby splňoval nastavené podmínky.

Dalším omezením, které souvisí i s tím, že se má jednat výhradně o klientskou aplikaci, je nemožnost využití vzdálených služeb pro zpracování nebo úpravu dat (například pro uložení dat, vyhodnocení výrazů apod.). Veškerá komunikace musí probíhat v „ověřené zóně“ aplikace a hlavní jádro aplikace musí být funkční i bez konektivity do sítě Internet. Výjimkou jsou komponenty, které komunikaci s externím serverem explicitně vyžadují.

2.4 Aktualizace použitých technologií

Protože správce rozložení stránky používá externí technologie a nástroje, které se neustále vyvíjejí, je třeba provést jejich aktualizace na nejnovější dostupné verze. Týká se to zejména knihovny Ext JS a textového editoru TinyMCE, o kterých se zmiňuje další text.

2.5 Začlenění nového kódu do existující aplikace

Nově vytvořený kód musí být formou rozšíření (plugin) umístěn v samostatných souborech tak, aby správce rozložení stránky bylo možné provozovat v závislosti na nastavení s i bez jeho zapojení. Existující kód správce rozložení je možné upravit, nicméně pouze v obecné rovině a veškerý specifický kód týkající se výhradně podpory Gadgets API musí být umístěn samostatně.

Je možné upravovat i další soubory, ze kterých se správce rozložení stránky sestává (HTML šablony, CSS – kaskádové styly i obrázky, je-li to nutné).

Kód nesmí obsahovat žádné pevně nastavené textové zprávy, které slouží k informování uživatele. Správce rozložení stránky je totiž vícejazyčný a lokalizované texty má umístěné v samostatných souborech, do kterých musí být umístěny i všechny nově vytvořené zprávy. Zprávy je nutné psát česky a případně anglicky; zajistit překlad ale není nezbytně nutné.

KAPITOLA 3

Analýza správce rozložení stránky projektu KiWi

Při analýze správce rozložení stránky projektu KiWi bylo čerpáno z dokumentace ([27]), konzultací s jeho autorem a vlastních poznatků.

Správce rozložení stránky (angl. layout manager) projektu KiWi je webová aplikace, která umožňuje uživatelům rozdělit okno prohlížeče na několik oblastí, přičemž každá z těchto oblastí může obsahovat komponenty s odlišným obsahem.

Jedná se o systém implementovaný v prostředí webového prohlížeče pomocí technologií HTML, CSS a JavaScript. Poslední zmiňovaná technologie – JavaScript – hraje ve správci významnou úlohu, protože zajišťuje nejen interakci s uživatelem, ale i samotné načtení a sestavení stránky. Můžeme tedy říci, že správce rozložení stránky je JavaScriptovou aplikací v prostředí webového prohlížeče.

Pro jeho implementaci využili jeho tvůrci javascriptový framework Ext JS[1], který je kromě frameworku znám i jako nástroj pro tvorbu tzv. RIA (*Rich Internet Applications*) aplikací, tedy aplikací, které se „*snaží překlenout rozdíly mezi klasickou webovou aplikací a desktopovou aplikací a snaží se v rámci webového prohlížeče napodobovat desktopové aplikace svým vzhledem i chováním, a poskytnout tak vyšší uživatelský komfort*“ [24]

Na Internetu lze nalézt systémy, které jsou svým chováním správci rozložení stránky projektu KiWi podobné. Mezi nejznámější nejspíše patří iGoogle společnosti Google[2] nebo My Yahoo! společnosti Yahoo[3]. Podobný koncept ale v poslední době převzaly i další portály, včetně britské BBC[4] nebo českého Seznam.cz[5].

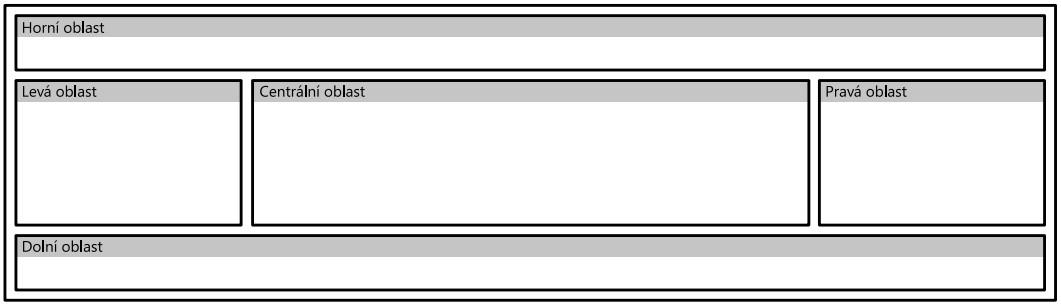
3.1 Struktura správce rozložení stránky

Správce rozložení stránky na základě své konfigurace sestaví pracovní plochu, která je rozdělena do několika oblastí (typicky pěti – horní, pravá, dolní, levá a centrální, obrázek 3.1). V každé z těchto oblastí může být libovolné množství sloupců, které danou oblast dále člení. V každém sloupci pak může být libovolné množství pod sebou umístěných komponent, přičemž komponentou zde rozumíme zdrojově, funkčně a vizuálně oddělenou aplikaci menšího rozsahu (obrázek 3.2). Jednotlivé komponenty jsou, stejně jako vlastní správce rozložení stránky, tvořeny kombinací jazyků HTML, CSS a JavaScript. Každá z nich má svůj (v rámci správce rozložení stránky) jedinečný identifikátor, vlastní konfiguraci a obsah.

Rozvržení stránky je plně konfigurovatelné, jednotlivé oblasti lze v závislosti na nastavení zmenšovat/zvětšovat, skrývat či úplně zavírat. Vlastní komponenty lze myší pomocí metody drag&drop přesouvat nejen v rámci jedné oblasti, ale i mezi nimi. Každá oblast, stejně tak

[1] Webovou prezentaci frameworku Ext JS lze nalézt na http://www.extjs.com
[2] http://www.google.com
[3] http://www.yahoo.com
[4] http://www.bbc.co.uk
[5] http://www.seznam.cz

jako každá z komponent, je jasně identifikována textovým popiskem v záhlaví, které je s danou oblastí či komponentou pevně spojeno a které usnadňuje orientaci uživatele.



Obrázek 3.1: Výchozí oblasti ve správci rozložení stránky

Obrázek 3.2: Umístění komponent ve správci rozložení

Cílem této práce je implementovat podporu pro komponenty typu Google Gadgets.

3.2 Sestavování pracovní plochy

Konfigurace rozvržení stránky i konfigurace a obsah všech komponent jsou umístěny v jednom jediném XML souboru. Strukturu tohoto XML lze nalézt v dokumentaci. Při otevření stránky ve webovém prohlížeči dochází k vytvoření instance observeru (úloha observeru je popsána v další části) a správce rozložení stránky, který provede inicializaci dalších struktur. Správce rozložení stránky následně pomocí AJAX požadavku získá ze serveru zmíněný konfigurační XML soubor, který zpracuje, a na jeho základě sestaví pracovní plochu včetně obsažených komponent.

Sestavení pracovní plochy probíhá ve dvou hlavních fázích:

1. sestavení jednotlivých oblastí,
2. vložení komponent do správných oblastí.

Při načítání XML a analýze komponent v něm uložených dochází v jejich obsahu k nahrazení některých speciálních proměnných. Hodnoty těchto proměnných se získají z aktuálního stavu správce rozložení nebo z konfigurace prostředí či komponenty. Mezi tyto proměnné patří skutečný identifikátor komponenty, hodnoty uživatelských zpráv (v závislosti na nastaveném jazykovém prostředí) nebo například uživatelské nastavení. Například `__MY_ID__` je nahrazeno za skutečný identifikátor dané komponenty, `__MSG_custom_message__` za uživatelskou zprávu a `__UP_custom_variable__` za hodnotu nastavení, které může ovlivňovat sám uživatel pomocí speciálního formuláře pro nastavení.

3.3 Způsob komunikace mezi komponentami

Správce rozložení poskytuje několik metod, kterými komponentám umožňuje komunikovat s okolím i s ním samotným. Jejich kompletní soupis lze nalézt v dokumentaci správce rozložení nebo v [27]. Mezi základní metody patří získání reference na vybranou komponentu, její umístění na stránce, hodnotu uživatelské proměnné nebo lokalizovaný text. Důležitým prvkem, který správce rozložení využívá, je prvek observer, který implementuje stejnojmenný návrhový vzor (návrhový vzor observer zajišťuje propagaci událostí určitého typu všem závislým objektům [37]). Jednotlivé komponenty tak jsou schopné reagovat na probíhající změny, přestože jich samotných se dané změny třeba ani netýkají. Typickým příkladem propagace události může být změna pozice komponenty nebo její uzavření, ale i programově či uživatelem cíleně vyvolaná změna konkrétní vlastnosti. Každá komponenta tak má přehled nejen o svém vlastním stavu, ale i o celém prostředí, ve kterém běží.

3.4 Začlenění komponent do správce rozložení

Jak již bylo zmíněno, jednotlivé komponenty jsou vytvořené v jazycích HTML, CSS a JavaScript. Jejich vložení do prostředí správce rozložení je realizováno začleněním jejich kódu do patřičné větve DOM správce rozložení. Nejedná se tedy o oddělené aplikace, protože se správcem rozložení sdílejí DOM dokument, kaskádové styly (CSS) i globální prostor JavaScriptu. Je tak teoreticky možné, že jedna nevhodně vytvořená komponenta způsobí pád celého správce rozložení.

V předchozím textu bylo rovněž zmíněno, že konfigurace i obsahy komponent se načítají z jednoho XML zdroje. Za výhodu proto lze považovat, že správce rozložení stránky používá strukturu komponent částečně kompatibilní se strukturou Google Gadgets [27, 21].

3.5 Stávající podpora Google Gadgets

Současná verze správce rozložení stránky má již některé části Gadgets API implementovány. Tyto metody jsou pevně zakomponované do jádra správce a bude nutné je přesunout

do samostatného modulu, případně celé znovu přepsat.

V uvedené verzi probíhá zpracování „obyčejných“ komponent i komponent typu Google Gadgets podobně. Liší se pouze ve zpracování vlastností specifických pro gadgety a v simulaci některých rysů Gadgets API, které dosud nejsou nativně podporovány. Správce rozložení stránky totiž neumožňuje identifikovat danou komponentu za běhu, proto při prvotní analýze jejího obsahu provádí v načteném těle řetězcové nahrazování všech obecných funkcí (typu `call(getMyId())`) za odpovídající konkrétní hodnotu (např. `call(1567)`). Protože Gadgets API obsahuje podobných funkcí více, je současné řešení nevyhovující, bude nutné jej změnit a komponenty identifikovat, až když o to skutečně požádají.

Mezi podporované části Gadgets API patří podpora pro uživatelská nastavení (`gadgets.Prefs`), základní pomocné funkce (`gadgets.util`) a uživatelské zprávy (`gadgets.MiniMessage`). Kompletní popis Gadgets API je uveden v kapitole 4.

3.6 Serverová část

Jedním z požadavků na tuto práci je, že se musí jednat o aplikaci běžící na straně uživatele. Správce rozložení stránky toto kritérium splňuje, nicméně některé jednoduché skripty na straně serveru využívá. Jedná se pouze o tři soubory, které jsou vytvořené v programovacím jazyce PHP (*PHP: Hypertext Preprocesor*).

KAPITOLA 4

Analýza Google Gadgets

Při analýze Google Gadgets bylo čerpáno ze specifikace ([19], [14]) a vlastních poznatků.

Gadgets jsou jednoduché HTML a JavaScriptové aplikace, které mohou být vloženy do webových stránek a jiných aplikací [19]. Umožňují vývojářům jednoduše psát užitečné webové aplikace, které budou fungovat kdekoli na webu bez nutnosti úprav [14]. Mezi typické aplikace tohoto typu lze zařadit výpis nejnovějších článků na blogu, předpověď počasí nebo přehrávač videa.

Gadgety může vytvořit prakticky kdokoli, kdo dodržuje platnou specifikaci[1]. Specifikace je obecná, nicméně v této práci se zaměříme hlavně na implementaci společností Google, Inc.[2] Vytvořené gadgety mohou jejich autoři umístit do adresáře, který společnost Google provozuje[3]. V něm uložené gadgety pak mohou uživatelé procházet, hodnotit a vkládat do svých webových stránek.

4.1 Vkládání Google Gadgets

Google Gadgets lze vložit do své vlastní webové stránky nebo do prostředí, které Gadgets API podporuje. Oba postupy jsou poměrně snadné a určené i pro méně zkušené uživatele.

Vložení do vlastní stránky je velmi jednoduché. Stačí si v adresáři gadgetů najít vhodný gadget a pomocí připraveného průvodce si nechat vygenerovat stručný HTML kód, který je třeba do existující webové stránky vložit. Tento HTML kód obsahuje pouze jeden element `<script>`, který odkazuje na servery společnosti Google a který informace o vkládaném gadgetu obsahuje zakódované ve své URL adrese.

Po vložení elementu `<script>` do webové stránky dojde při každém jejím načtení k jeho načtení a vykonání, kdy je na dané místo dokumentu pomocí javascriptové metody `document.write` vložen nový, dynamicky sestavený HTML kód obsahující tabulku (`table`) a v ní vložený plovoucí rám (`iframe`). Tabulka má pouze grafický účel (zajišťuje vykreslení ohraničení, záhlaví a zápatí gadgetu), zato plovoucí rám zajišťuje veškerou funkčnost gadgetu. Odkazuje sice opět na servery společnosti Google (podobně jako dříve zmíněný element `script`), tentokrát ovšem na službu, která již zajistí zpracování a zobrazení daného gadgetu.

Vložení do prostředí, které Gadgets API podporuje, je obvykle ještě jednodušší. Na tomto místě zmiňme službu *iGoogle*, jež, jak je z názvu patrné, je provozována společností Google. Jedná se o personalizovanou osobní stránku, která se registrovaným uživatelům zobrazuje místo obvyklé úvodní stránky vyhledávače. Je-li uživatel registrován, je mu již při procházení adresáře gadgetů zobrazen speciální odkaz, jehož navštívením je gadget na jeho personalizovanou stránku automaticky vložen. Podobné odkazy nabízí i další poskytovatelé, případně uživatelům dávají k dispozici jednoduchý formulář, do kterého stačí pouze zkopírovat adresu daného gadgetu.

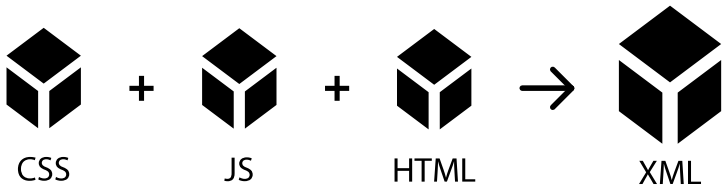
[1] Specifikaci lze nalézt v [14]

[2] Popis specifikace od společnosti Google lze nalézt na http://code.google.com/intl/cs-CZ/apis/gadgets/docs/reference/

[3] Adresář gadgetů lze najít na http://www.google.com/ig/directory

4.2 Struktura modulu

Modulem rozumějme XML soubor obsahující definici jednoho konkrétního gadgetu. Součástí této definice je jednak konfigurace daného gadgetu určující jeho vlastnosti v prostředí, ve kterém běží, a jednak vlastní kód aplikace, kterou gadget představuje (obrázek 4.1). Struktura tohoto XML souboru je pevně určená specifikací [21] a nelze ji měnit. Jak již bylo zmíněno v Začlenění komponent do správce rozložení na straně 8, struktura XML dle specifikace je částečně kompatibilní se strukturou, kterou používá správce rozložení stránky, což nám usnadní jeho načítání.



Obrázek 4.1: Složení gadgetu

4.2.1 Ukázka struktury XML

```
<Module>
 <ModulePrefs title="Developer Forum" height="200" author="Jane Smith">
   <Require feature="dynamic-height"/>
   <Optional feature="shareable-prefs"/>
 </ModulePrefs>
 <UserPref name="difficulty" display_name="Difficulty"
    datatype="enum" default_value="4">
   <EnumValue value="3" display_value="Easy"/>
   <EnumValue value="4" display_value="Medium"/>
   <EnumValue value="5" display_value="Hard"/>
 </UserPref>
 <Content>
    obsah gadgetu ve formátu HTML
 </Content>
</Module>
```

Ukázkový kód obsahuje požadavek na nutnou (`<Require>`) a na volitelnou (`<Optional>`) podporu modulů, uživatelské nastavení (`<UserPref>`) a vlastní obsah (`<Content>`).

Části `<Require>` a `<Optional>` jsou nezbytné pro připojení patřičných javascriptových knihoven; způsobu zpracování XML se věnuje další část textu. Část `<UserPref>` se využívá pro definování uživatelských proměnných, které může prostřednictvím speciálního formuláře pro nastavení měnit sám uživatel.

Gadget může obsahovat více částí `<Content>`, protože prostředí, v němž běží, může podporovat více způsobů jeho zobrazení (malé okno, velké okno, ...). Pro každé zobrazení je pak možné mít odlišný obsah gadgetu.

Dalším prvkem, který může být v části `<ModulePrefs>` obsažen, je prvek `<Locale>` s atributy `lang` a `country`. V něm jsou umístěné uživatelské zprávy daného jazyka a v závislosti na zvoleném nastavení jsou hodnoty těchto zpráv dosazeny do proměnných použitých v těle gadgetu. Lze tak vytvářet vícejazyčné aplikace, které se přizpůsobují nastavení prostředí nebo volbě uživatele.

Kompletní XSD (*XML Schema Definition*[4]) lze nalézt jako přílohu *A* specifikace[5].

4.3 Způsob zpracování gadgetu

Při zpracovávání gadgetu dochází dle specifikace k těmto činnostem:

1. načtení požadovaného XML modulu
2. analýza modulu a HTTP požadavku
3. nahrazení speciálních proměnných (*hangman variables*) v obsahu gadgetu za skutečné hodnoty
4. vložení požadované části obsahu gadgetu (HTML kódu) do připravené šablony a připojení všech vyžadovaných javascriptových knihoven (knihovny jádra doplněné o knihovny explicitně vyžádané – pomocí `<Require>` a `<Optional>`)
5. sestavení výsledné HTML stránky s gadgetem a odeslání klientovi

Veškerá činnost zpracování, analýzy a vyhodnocení XML tedy probíhá na serveru, klientovi je předán až HTML kód připravený k okamžitému vykreslení. Podstatné také je, že o zpracování požadavků se starají servery poskytovatele prostředí (např. společnosti Google) a autor gadgetu nemá žádné možnosti konkrétní požadavek ovlivnit. Existuje sice možnost, kdy obsah gadgetu není součástí XML a načítá se z externího umístění (jeho autor tedy požadavek teoreticky ovlivnit může), nicméně v tom případě nemůže využívat žádné části Gadgets API, a proto tuto variantu nebudeme dále uvažovat[6].

Protože obsah gadgetu je statický, může gadget pro interaktivní běh využít pouze klientské skriptování – JavaScript. Proto je součástí specifikace i sada javascriptových funkcí, které poskytují podporu pro komunikaci, práci s daty a tvorbu základních prvků uživatelského rozhraní.

[4] XSD popisuje strukturu XML dokumentu

[5] http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/Gadgets-API-Specification.html#gadgetsExtendedXSD

[6] Ve specifikaci je tento typ zpracování uveden jako *Type URL*.

4.4 Javascriptové Gadgets API

Javascriptové API využívá globální jmenný prostor `gadgets` a je rozdělené na dvě části:

1. *jádro (Core API)* a
2. *specifické moduly (Feature-Specific API).*

Implementace všech funkcí jádra je dle specifikace povinná a prostředí, v němž gadget běží, je musí podporovat. Naopak podpora specifických modulů je volitelná a chce-li gadget některý z těchto modulů použít, musí o něj výslovně požádat (pomocí `<Require>` v XML, viz Struktura modulu, str. 11). Pokud požadovaný modul prostředí nepodporuje, je zpracování gadgetu odmítnuto [14].

Toto rozdělení funkcí má svůj význam i v množství přenášených dat – klientovi se totiž přenášejí pouze ty knihovny, které jsou pro běh daného gadgetu skutečně potřebné.

4.4.1 Jádro (Core API)

- *Prefs*
  uživatelská nastavení
- *io*
  komunikace se vzdáleným serverem
- *json*
  podpora pro formát JSON
- *util*
  funkce obecného charakteru

4.4.2 Specifické moduly (Feature-Specific API)

- *MiniMessage*
  jednoduché zprávy pro informování uživatele
- *TabSet*
  správce záložek (tabs) a jejich obsahu
- *flash*
  podpora pro jednoduché vkládání Adobe Flash animací
- *pubsub*
  model publisher-subscriber
- *rpc*
  vzdálené volání procedur
- *skins*
  podpora přizpůsobení vzhledu gadgetů prostředí, ve kterém běží

- *views*
  různé obsahy gadgetu v závislosti na zvoleném módu zobrazení
- *window*
  funkce pro práci s oknem gadgetu

Bohužel, v některých směrech není specifikace zcela jednoznačná, a proto je implementace daných metod komplikovaná.

4.5 Verze specifikace

Gadgets API je v současné době součástí projektu OpenSocial, který definuje jednotné API pro přístup k datům umístěným na sociálních sítích [23]. Úkolem tohoto projektu nicméně je implementovat podporu Gadgets API (nikoliv OpenSocial API), proto se OpenSocial dále nebudeme zabývat[7].

Tato práce vychází ze specifikace verze 0.9[8]. V březnu 2010 byla sice vydána nová verze specifikace (1.0[9]), nicméně na její zapracování v rámci této práce již nebylo dostatek prostoru. Popis API, ze kterého bylo při tvorbě vycházeno, lze nalézt v [18] a [21].

4.6 Legacy API

Postupem vývoje specifikace došlo ke změně strategie a vytvoření objektové koncepce API s jedním jediným globálním objektem. Nicméně kvůli zachování zpětné kompatibility je nutné poskytovat podporu i pro předchozí řešení. Rozhraní jednotlivých metod se většinou nezměnilo, takže stačí provést mapování původního rozhraní na rozhraní nové ([20]).

[7] Podrobnější informace o OpenSocial lze případně nalézt na http://www.opensocial.org/ nebo v [23].
[8] http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/Gadgets-API-Specification.html
[9] http://opensocial-resources.googlecode.com/svn/spec/1.0/OpenSocial-Specification-Release-Notes.xml

KAPITOLA 5

Použité technologie a standardy

Správce rozložení stránky je vytvořen pomocí technologií HTML, CSS a JavaScript [27]. Stejně tak jsou tyto technologie využity v prostředí Gadgets API [19]. Jejich použití pro další rozšiřování se tak jeví jako vhodné, ne-li nutné.

Pokud bychom chtěli využít jiné technologie (Flash[1], Silverlight[2]) narazíme na dva hlavní problémy:

1. podpora těchto technologií není dle specifikace zaručena,
2. bylo by nutné implementovat vykreslovací jádro HTML s podporou CSS (tedy simulovat funkci webového prohlížeče).

Implementace vykreslovacího jádra by byla sice zdlouhavá, ale možná. Nicméně nejistota podpory zmíněných technologií je nepřijatelná, a tak zmíněné alternativní řešení použít nelze.

5.1 HTML

HTML (*HyperText Markup Language*) je značkovací jazyk určený pro tvorbu webových stránek. Objevil se na počátku 90. let 20. století aplikací jazyku SGML (*Standard Generalized Markup Language*). Stručný pohled do historie HTML lze nalézt například na webu konsorcia W3C [10]. Nejnovější standardizovanou verzí tohoto jazyka je verze 4.01 z prosince 1999, jejíž specifikaci lze nalézt taktéž na webu konsorcia W3C.

V současné době (jaro 2010) je ve fázi návrhu HTML5 [13], a přestože proces standardizace bude ještě několik let trvat [6], vybrané části budoucího standardu již jsou v některých webových prohlížečích podporovány [7] a lze je využít.

V roce 2004 se objevil nový standard XHTML, který oproti HTML 4.01 zavádí některé změny, zejména v typu dokumentu. XHTML dokumenty jsou založené na jazyce XML 1.0 (*Extensible Markup Language*), a musí tak dodržovat přísnější pravidla, která se standardem XML souvisí [11]. Protože v našem případě budeme přebírat obsah již existujících gadgetů, je lepší variantou využít HTML 4.01, které je méně striktní [11]. Pojmem *dynamické HTML* (neboli *DHTML*) chápeme spojení technologií HTML, CSS, DOM a JavaScript [5].

5.2 XML

XML (*eXtensible Markup Language*) je univerzální značkovací jazyk vyvinutý a standardizovaný konsorciem W3C. V době psaní tohoto textu je jeho nejnovější verzí verze 1.0 (pátá edice)[3] [12]. V kontextu této práce lze XML chápat hlavně jako prostředek pro serializaci dat.

[1] http://www.adobe.com/products/flash/
[2] http://www.silverlight.net/
[3] Specifikaci XML lze nalézt v [12]

5.3 JSON

Formát JSON (*JavaScript Object Notation*) navrhl Douglas Crockford a je popsán v RFC 4627[4]. Jedná se o jednoduchý datový formát založený na podmnožině syntaxe JavaScriptu – na polích a objektech [42]. Podobně jako XML je i formát JSON určen pro výměnu dat, na rozdíl od XML je ale více kompaktní.

5.4 DOM

DOM (*Document Object Model*) představuje způsob, jak pracovat s objekty v rámci dokumentu. V kontextu této práce chápeme DOM jako objektově orientovanou stromovou reprezentaci HTML a XML dokumentů. Pro práci s DOM se v rámci webového prohlížeče využívá skriptovací jazyk JavaScript.

Jedná se o nejrozšířenější metodu podporovanou v běžných webových prohlížečích, která je velmi flexibilní [38]. Protože dynamické HTML používá pro manipulaci s HTML dokumentem právě DOM, jedná se o jedno ze stěžejních rozhraní použitých v této práci.

Ačkoli nejnovější verze standardu (DOM level 3) je již z roku 2004 [1], jeho podpora ve webových prohlížečích není příliš dobrá [4, 30] a v praxi je nutné používat specifikaci starší, DOM Level 2.

5.5 JavaScript

JavaScript je objektově orientovaný skriptovací jazyk hojně využívaný v prostředí webových stránek. Jedná se o jazyk slabě typovaný, založený na prototypech, který je nejčastěji interpretován na straně klienta[5]. V prostředí webového prohlížeče pracujeme s podmnožinou jazyka označovanou jako klientský JavaScript, který přináší některá omezení, ale zároveň poskytuje i některá rozšíření (globální objekt window, prostředky pro DOM operace nebo například prvky uživatelského rozhraní) [26].

Značka JavaScript je vlastněná společností Sun Microsystems (Oracle) [25], nicméně v poslední době se na vývoji jazyka podílí především Mozilla Foundation [3]. Poslední stabilní verzí je sice verze 1.8.1, nicméně většina dnešních prohlížečů podporuje mírně upravený ECMAScript 3[6], který odpovídá JavaScriptu verze 1.5 [22]. Prohlížeče z rodiny Mozilla jsou tak jediné, které nejnovější verze JavaScriptu podporují [39]. Proto jsou i v této práci využity vlastnosti verze 1.5.

5.6 CSS

CSS (*Cascading Style Sheets*) je jazyk pro definici vzhledu webových stránek. V současné době je standardizována pouze verze 2 z roku 1998, nicméně i podpora verze 2.1 (jejíž proces

[4] http://www.ietf.org/rfc/rfc4627.txt

[5] Existuje i serverová implementace, viz například http://ejohn.org/blog/server-side-javascript-with-jaxer/ nebo http://ejscript.org/

[6] Vývoj ECMAScript 4 byl pozastaven, specifikace ECMAScript 5 byla vydána v prosinci 2009.

standardizace stále probíhá) je v dnešních prohlížečích velmi dobrá [29, 31]. V době tvorby tohoto textu se pracuje na návrhu specifikace CSS 3 a podobně jako u návrhu HTML 5 jsou i některé části návrhu CSS 3 v moderních prohlížečích již podporované [29]. Specifikaci verze 2.1 lze nalézt na webu konsorcia W3C [2].

5.7 PHP

PHP (*PHP: Hypertext Preprocessor*, dříve *Personal Home Page*) je velmi oblíbeným [41] jazykem pro tvorbu dynamických webových stránek. První verzí, která nejvíce připomíná současné PHP, bylo PHP 3.0, které vydali Andi Gutmans a Zeev Suraski v roce 1997 [8]. Bližší informace o historii jazyka lze najít v [8].

Jedná se o slabě typovaný objektově orientovaný jazyk interpretovaný na serveru. Poslední dostupná verze v době psaní textu je verze 5.3.2 [9]. Více se lze o PHP dozvědět z oficiálního webu[7] nebo například z českého weblogu Jakuba Vrány[8].

5.8 Ext JS

Ext JS je javascriptová knihovna pro tvorbu tzv. RIA (*Rich Internet Applications*) aplikací[9]. Obsahuje bohatou knihovnu komponent pro tvorbu uživatelských rozhraní, práci s daty či komunikaci [16], čímž umožňuje snadnou tvorbu pokročilých aplikací a odstiňuje vývojáře od nutnosti zabývat se specifiky jednotlivých prohlížečů.

V době psaní tohoto textu je nejnovější dostupnou verzí verze 3.2 [16]. Více informací lze nalézt na oficiálním webu Ext JS[10].

5.9 AJAX

AJAX (*Asynchronous JavaScript and XML*) není ani technologie ani standard, je to spíše přístup k otázce interakce mezi klientem a serverem. Tento přístup je založen na přenosech malého množství dat na server a získání patřičné odpovědi [42].

Nejedná se o nový přístup – existuje již od počátků dynamického HTML – ale pouze o nové pojmenování. Pojem AJAX publikoval ve svém článku Jesse James Garrett v únoru 2005 [28] a postupně ho převzali další vývojáři. Garrett pod pojmem AJAX chápe spojení technologií XHTML, CSS, DOM, XML a XSLT[11] v kombinaci s objektem XMLHttpRequest[12] a JavaScriptem.

Historii, principům a technikám AJAXu se věnuje například [42].

---

[7] http://www.php.net

[8] http://php.vrana.cz/

[9] Definice pojmu *RIA* byla uvedena v Analýza správce rozložení stránky projektu KiWi na straně 6

[10] http://www.extjs.com/

[11] *eXtensible Stylesheet Language Transformations*, více informací lze nalézt například na http://interval.cz/clanky/kompletni-pruvodce-xslt-uvod-do-problematiky/

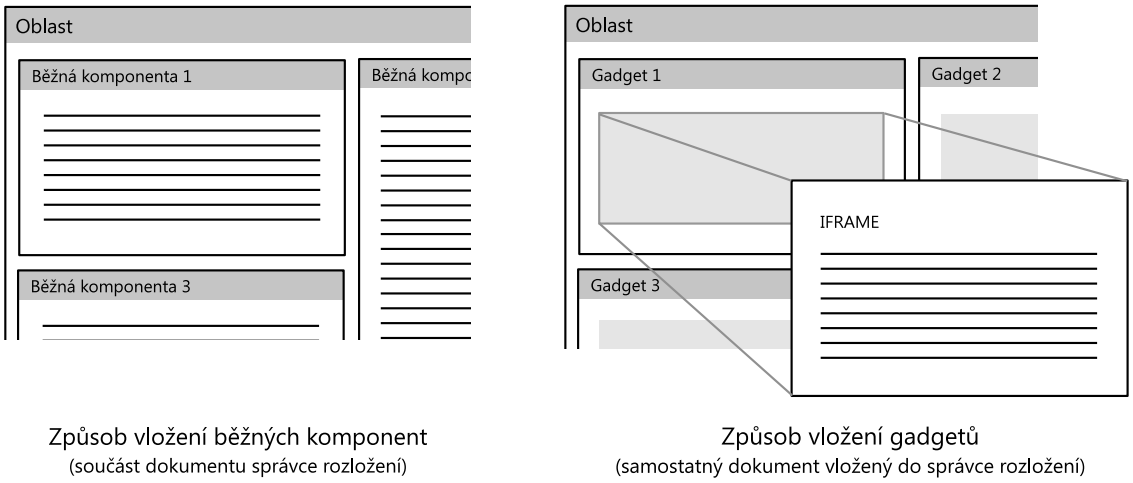
[12] Objekt XMLHttpRequest umožňuje navázat HTTP spojení pomocí JavaScriptu.

KAPITOLA 6

Návrh řešení

6.1 Možnost vložení obsahu gadgetu do správce rozložení

Současné řešení vkládá kompletní obsah jednotlivých komponent přímo do DOM správce rozložení, a komponenty tak se správcem rozložení sdílejí DOM, CSS i globální objekt JavaScriptu[1]. Tento přístup se u gadgetů, jejichž obsah není tvořen přímo na míru, ale je převzat z již vytvořených aplikací, jeví jako nevhodný. Lze totiž předpokládat, že gadgety obsahují vlastní CSS pravidla nebo javascriptové proměnné, které by mohly při začlenění do správce rozložení způsobit problémy (kolize jmen, narušení kaskády CSS pravidel apod.). Nejvhodnějším řešením se proto zdá být vytvoření samostatného dokumentu pro každý z gadgetů. Tento dokument je následně vložen do správce rozložení jako celek – jediný prvek (obrázek 6.1).



Obrázek 6.1: Způsob začlenění gadgetu do správce rozložení

6.1.1 Možnosti oddělení obsahu gadgetu od správce rozložení

V HTML existují dvě možnosti, jak začlenit jeden dokument do druhého: pomocí `<object>` nebo pomocí `<iframe>`. Řešení s `<object>` příliš známé není, přestože ho přímo zmiňuje specifikace[2]. Nevýhoda tohoto řešení spočívá v jeho podpoře mezi prohlížeči. Existují totiž různé způsoby, jak je nutné objekt vložit, aby byl správně interpretován [32]. Naproti tomu řešení s `<iframe>` je poměrně známé, hojně používané a odzkoušené. Přestože zejména kvůli problémům s přístupností je snaha od používání rámců upouštět, nenabídlo dosud HTML

[1] Blíže je tento postup popsán v Začlenění komponent do správce rozložení na straně 8.
[2] http://www.w3.org/TR/REC-html40/struct/objects.html#h-13.5

vyjma `<object>` žádnou alternativu. Proto se především díky bezproblémové podpoře mezi prohlížeči zdá být `<iframe>` lepší volbou.

Při sestavování stránky lze tyto `iframe` prvky vytvářet dynamicky a vkládat je na místa, na která obsah gadgetu patří. Vlastní obsah gadgetu je pak nutné vložit do vytvořeného iframe prvku, nikoliv přímo do stránky se správcem rozložení.

6.1.2 Vložení obsahu gadgetu do iframe

Při dynamickém vytváření prvků `iframe` lze vytvořit prázdný `iframe` (obsahující prázdný dokument), s nímž lze dále libovolně pracovat. Protože ale bude nutné do něj kromě vlastního obsahu gadgetu pokaždé umístit i vazby na další knihovny (ať už javascriptové, nebo CSS), bude vhodnějším řešením vytvořit speciální HTML šablonu, která již tyto vazby bude obsahovat a do které bude stačit vložit pouze vlastní obsah gadgetu. Správce rozložení pak nebude vytvářet prvky `iframe` prázdné, ale obsahující tuto šablonu, čímž si vkládání obsahu gadgetu značně zjednodušíme.

Obsah gadgetu získáme z XML jako řetězec (string) obsahující HTML kód. Proto nejjednodušším způsobem, jak tento kód vykonat, je přiřadit jej přes vlastnost `innerHTML` do těla (`body`) vytvořeného `iframe` prvku.

6.2 Způsob identifikace komponent

Správce rozložení stránky v současné době neumožňuje provést identifikaci komponenty v průběhu běhu aplikace. Je proto nutné navrhnout způsob, který tuto identifikaci umožní.

Každé komponentě je při načítání a analýze zdrojového XML souboru s komponentami přidělen jedinečný identifikátor, který je po celou dobu běhu aplikace neměnný. Použijeme-li pro vložení gadgetu do správce rozložení elementy `iframe`, dokážeme z dokumentu uvnitř `iframe` pomocí JavaScriptu jednoduše přistoupit k dokumentu se správcem rozložení (vycházíme z toho, že oba dokumenty jsou umístěny v rámci stejné domény). Proto stačí, aby si správce rozložení tyto identifikátory ukládal a spojoval s konkrétními instancemi `iframe` elementů. Poté se stačí dotázat na konkrétní instanci prvku `iframe` a na základě uložené hodnoty jeho identifikátor získat. Tímto způsobem je gadget schopen získat své identifikační číslo kdykoli během svého běhu.

6.3 Rozšíření správce rozložení

Jak již bylo zmíněno v Stávající podpora Google Gadgets, správce rozložení v současné verzi již částečnou podporu pro prvky gadgets poskytuje. Některé části specifikace nicméně podporovány nejsou a je třeba navrhnout jejich začlenění.

6.3.1 Vyvolání události informující o dokončení načítání

Jakmile je načítání všech komponent dokončeno a stránka je sestavena, vyvolá správce rozložení pomocí observeru událost `layoutReady` informující všechny komponenty, že stránka

byla sestavena a je připravena k běhu. Při začlenění gadgetů (v prvcích `iframe`) ale může dojít k problémům, kdy tato událost může být vyvolána, přestože vlastní obsahy gadgetů dosud načtené nebyly. Načítání obsahu iframe totiž probíhá asynchronně a správce rozložení nemá přesné informace, v jakém stavu se toto načítání nachází.

Tedy v případě, kdy se ve stránce nějaké gadgety nacházejí, bude nutné kontrolovat jejich stav a událost `layoutReady` vyvolat, až když jsou skutečně načtené. Tento stav není možné kontrolovat pomocí standardní události `onload`, protože ta by byla vyvolána po načtení šablony gadgetu (nikoliv jeho obsahu), ale bude nutné ho kontrolovat programově, a to ověřováním, zda již byly dokončeny všechny akce nutné k běhu daného gadgetu.

Podobný význam jako událost `layoutReady` má i metoda `gadgets.util.registerOnLoadHandler`. Ta gadgetu umožňuje zaregistrovat funkce, které se mají provést okamžitě po načtení jeho dokumentu. A protože v našem případě načtením dokumentu chápeme načtení celého správce rozložení, budou veškeré události zaregistrované pomocí `registerOnLoadHandler` vykonány po přijetí události `layoutReady`.

6.3.2 Jazykové lokalizace

V současné době podporuje správce rozložení stránky vkládání lokalizovaných zpráv pouze v jednom jazyce, a to přímo do XML zdroje. Naproti tomu specifikace gadgetů umožňuje vložení více jazyků, jejichž obsah může být umístěn nejen přímo v XML zdroji, ale i v externích souborech [17]. Jednotlivé jazykové balíky lze navíc blíže rozlišit podle státu, a mít tak například rozdílné anglické verze pro Velkou Británii a pro Spojené státy americké.

Aplikování jednotlivých jazykových balíků probíhá postupně od konkrétních k obecným. Pokud tedy máme jako jazyk nastavenou angličtinu a jako stát Velkou Británii, probíhá hledání textu jazykové zprávy nejdříve v anglické verzi pro Velkou Británii, není-li nalezena, je hledána v anglické verzi pro všechny státy, a není-li nalezena ani v ní, je hledána ve výchozích zprávách pro všechny jazyky a všechny státy.

Protože naše řešení musí využívat technologie na straně klienta, bude nutné získávat jednotlivé jazykové balíky pomocí asynchronních AJAX požadavků. A jelikož vyhodnocení balíků musí proběhnout v pevně stanoveném pořadí, které není při asynchronních požadavcích garantováno, je nutné vyčkat na stažení všech balíků a až poté vyhodnocení provést.

Jelikož jsou jazykové balíky pevnou součástí gadgetu (gadget lze spustit až po jejich vyhodnocení), je nutné i v případě načítání externích jazykových zpráv vyvolání dříve zmíněné události `layoutReady` pozdržet.

6.3.3 Nastavení vlastností

Specifikace umožňuje pro jednotlivé vlastnosti (features) definovat vlastní parametry, jejichž další využití není ze specifikace zcela zřejmé. Tyto parametry lze definovat přímo v XML zdroji gadgetu a později načíst voláním funkce `gadgets.util.getFeatureParameters`. Proto bude nutné provést úpravu načítání XML tak, aby správce rozložení uschovával i obsah těchto parametrů. Jejich uschování je snadné, stačí pro každou z vlastností zavést asociativní pole, kde klíčem bude název vlastnosti a hodnotou načtená data.

6.3.4 Kontrola podpory vlastností

Součástí definice gadgetu je i seznam vlastností (features), které gadget pro svoji činnost vyžaduje. Tyto požadavky je nutné již při analýze gadgetu ověřovat, a pokud je není správce rozložení stránky schopen splnit, je nutné zpracování gadgetu pozastavit a informovat uživatele, že daný gadget nemůže být z důvodu nepodporování vykreslen.

6.4 Možnosti Gadgets API – jádro (Core API)

6.4.1 Prefs – uživatelská nastavení

Podpora pro uživatelská nastavení již je v rámci správce rozložení stránky zahrnuta a není třeba ji výrazně upravovat. Je pouze nutné zavést podporu lokalizace a upravit způsob identifikace komponent.

6.4.2 IO – komunikace se vzdáleným serverem

Funkce ze sady pro komunikaci umožňují získat obsah vzdálené URL adresy. Pro vzdálenou komunikaci bez nutnosti obnovení celé stránky se v dnešní době využívají tři hlavní techniky: komunikace pomocí skrytých plovoucích rámců (iframe), JSON-P (*JSON with padding*) nebo objekt XMLHttpRequest. Každá z nich má své výhody i nevýhody.

Metody komunikace

Metoda skrytých rámců je vhodná pro přenos dat na server, umožňuje totiž odeslat velké množství dat včetně obsahu souborů (což ostatní zmíněné přístupy neumějí). Komplikací je čtení odpovědi serveru, kterou je z bezpečnostních důvodů možné získat pouze tehdy, pokud komunikujeme v rámci stejné domény (tj. v rámci domény, na které běží správce rozložení stránky). Což je v našem případě problém, protože lze předpokládat, že gadgety budou chtít komunikovat zejména s externími doménami.

JSON-P funguje na principu dynamického vytváření prvků `<script>` a jejich vkládání do dokumentu. Cílovou adresou těchto prvků je vzdálená služba, které je v URL parametru předán název předem připravené javascriptové obslužné funkce (callback). Jakmile je vzdálený dokument stažen, je tato obslužná funkce vykonána a stažená data jsou předána v jejím argumentu. Podrobnější popis techniky lze najít například v [36]. Výhoda této techniky spočívá v možnosti provádět i mezidoménovou komunikaci. Ovšem na druhou stranu velkou nevýhodou je skutečnost, že vzdálená strana musí JSON-P podporovat a vrátit předem zformátovaná data (s čímž v našem případě počítat nemůžeme). Navíc nelze provádět HTTP požadavky pomocí metody POST – jedinou podporovanou metodou je metoda GET, která umožňuje předat pouze omezené množství parametrů. A přestože je podpora metody POST dle specifikace volitelná[3], její implementace by byla velmi vhodná.

[3] http://code.google.com/intl/cs-CZ/apis/gadgets/docs/reference/#gadgets.io.MethodType_field_detail

Poslední způsob – komunikace pomocí XMLHttpRequest – je v dnešní době pravděpodobně nejpoužívanější. Umožňuje vytvářet a odesílat HTTP požadavky libovolného typu a reagovat na změnu jejich stavu. Touto metodou nelze odesílat soubory (což v našem případě nijak nevadí), ovšem nelze ani provádět mezidoménovou komunikaci. Jediným možným řešením tohoto omezení je vytvoření jednoduchého proxy serveru, který poběží na serveru společně se správcem rozložení stránky (poběží na stejné doméně) a který bude spojení s externími doménami zprostředkovávat. Kombinace XMLHttpRequest a proxy serveru je tak v našem případě nejlepším řešením.

Autorizace

Součástí IO je i způsob autorizace, který ovšem souvisí se dříve zmíněnou platformou OpenSocial, a tak jej není nutné implementovat.

Parametry proxy serveru

I samotná specifikace s tím, že bude nějaký proxy server nasazen, počítá. Proto zavádí některé možnosti, jak jej nastavit – např. určit dobu uchovávání dat v cache. V našem případě by měl být proxy server co nejjednodušší (protože poběží na serveru a naším cílem je vytvořit především klientskou aplikaci), proto tyto možnosti konfigurace nebudeme při jeho vývoji uvažovat.

Typy obsahu

Specifikace uvádí čtyři typy obsahu, které lze pomocí IO funkcí získat:

- DOM
  vrací DOM objekt, používá se pro načítání XML
- FEED
  vrací objekt reprezentující RSS nebo Atom zdroj
- JSON
  vrací objekt získaný z JSON dat
- TEXT
  vrací obyčejný text, používá se pro načítání HTML

Získání obsahů typu DOM a TEXT je snadné, stačí vhodně zpracovat vlastnosti objektu XMLHttpRequest. I typ JSON lze díky podpoře v knihovně Ext JS zpracovat jednoduše, ale typ FEED bude vyžadovat ruční analýzu.

V dnešní době patří v rodině RSS mezi nejpoužívanější verze RSS 2.0 (*Really Simple Syndication*) následovaná RSS 0.91 (zde zkratka znamená *Rich Site Summary*) [42]. Obě verze se od sebe příliš neliší, a implementujeme-li podporu pro jednu verzi, poskytneme tím podporu i pro verzi druhou. Další verze RSS 1.0 (*RDF Site Summary*) příliš rozšířená není [42]

a její podporu proto nebudeme uvažovat. Z rodiny zdrojů Atom je nejpoužívanější verzí verze 1.0, proto implementujeme i její podporu.

Výsledkem zpracování typu FEED má být objekt, který obsahuje informace o zdroji (feed) a zvolený počet záznamů. Bude tedy nutné procházet DOM staženého dokumentu, dle jeho typu a verze hledat odpovídající informace a kopírovat je do výsledného objektu. Ext JS nám pro procházení a analýzu DOM poskytuje dobrou podporu, až na podporu jmenných prostorů v XML (*XML namespaces*). Přestože rodina Atom používání jmenných prostorů povoluje, nejsme je bohužel schopni pomocí knihovny Ext JS zpracovat. Proto na základě dohody s českou částí týmu projektu KiWi byla podpora jmenných prostorů z Atom zdrojů vyřazena.

6.4.3 JSON – podpora pro formát JSON

Knihovna Ext JS obsahuje třídu `Ext.util.JSON`, která zahrnuje metody `encode` a `decode` pro převod mezi objektovou a řetězcovou reprezentací formátu JSON. Proto stačí v případě implementace podpory JSON provést pouhé mapování na tuto třídu.

6.4.4 Util – funkce obecného charakteru

Správce rozložení již některé funkce ze třídy util obsahuje. Bude nutné implementovat funkci `getFeatureParameters` pro získání parametrů vlastností (features) a rozšířit implementaci funkce `hasFeature` tak, aby akceptovala i nově implementované vlastnosti.

6.5 Možnosti Gadgets API – Specifické moduly (Feature-Specific API)

6.5.1 MiniMessage – jednoduché zprávy pro informování uživatele

Zprávy informující uživatele již jsou do správce rozložení začleněny, nicméně bude nutné provést jejich úpravu. Zobrazení zpráv je řešeno pomocí samostatného okna (komponenta `Ext.Window`), které je přemístitelné (tažením myši). Protože nově bude obsah gadgetu začleněn do iframe, současné řešení by způsobilo, že se okno se zprávou zobrazí pouze v rámci daného gadgetu, nikoliv celého správce rozložení. Proto bude nutné událost zobrazení okna se zprávou delegovat z gadgetu na správce rozložení.

Řešení pomocí samostatných oken není typické. Dle specifikace by se zprávy měly zobrazovat přímo v těle gadgetu, proto bude vhodné implementovat i tuto formu zobrazení. Přičemž v těle gadgetu se zprávy budou zobrazovat v případě, kdy bude uveden HTML kontejner, do kterého mají být vloženy.

6.5.2 TabSet – správce záložek (tabs) a jejich obsahu

Bude nutné vytvořit dvě třídy odpovídající specifikaci – TabSet a Tab – kde TabSet reprezentuje správce záložek a Tab jednu konkrétní záložku. Záložka je definována názvem, obsahem a funkcí zpětného volání, která je vyvolána při její aktivaci.

Pro implementaci modulu TabSet se výborně hodí komponenta TabPanel, která je součástí knihovny Ext JS a která téměř přesně splňuje požadovanou funkcionalitu.

Použití zmíněné komponenty má ovšem i svá úskalí – komponenta totiž nepodporuje některé požadované metody. Mezi ně patří zarovnání záložek (`alignTabs`, záložky jsou místo toho umístěny vždy vlevo) a prohození pořadí dvou záložek (`swapTabs`). Pokud bychom chtěli tyto dvě funkce podporovat, museli bychom rozšířit komponentu Ext JS (což v našem případě bohužel nelze pomocí veřejného API, ale museli bychom využít privátních struktur, jejichž význam může být v dalších verzích Ext JS změněn) nebo implementovat vlastní komponentu. Implementace vlastní komponenty samozřejmě poskytuje naprostou volnost, nicméně přináší i výrazné ztížení práce (návrh, vývoj, grafické rozhraní, napojení na Ext JS, testování apod.). A jelikož zmíněné metody nejsou z pohledu funkčnosti aplikace klíčové, bylo rozhodnuto, že je není nutné implementovat a bude použita standardní komponenta TabPanel.

6.5.3 Flash – podpora pro jednoduché vkládání Adobe Flash animací

Před vlastním vložením flash animace je nutné provést kontrolu, zda má uživatel Adobe Flash Player nainstalovaný a zda v dostatečné verzi. Pro detekci verze Flash Playeru lze využít Flash Player Detection Kit publikovaný přímo společností Adobe[4].

6.5.4 PubSub – model publisher-subscriber

Na principu odesílání a přijímání zpráv funguje komponenta observer, která je využívána pro komunikaci mezi komponentami. Při implementaci lze tedy využít jejích služeb a stačí provést mapování patřičných metod na rozhraní observeru.

6.5.5 RPC – vzdálené volání procedur

RPC (Remote procedure call) volání má v našem případě význam pouze mezi dvěma gadgety; volání mezi gadgetem a prostředím jsou nepodstatná, protože prostředí (správce rozložení stránky) žádná gadgets RPC volání nevytváří ani nepřijímá. V rámci rozšíření podpory pro RPC ale bude navržena i možnost, kdy pomocí RPC komunikuje gadget s v ním vnořeným plovoucím rámem (`iframe`), jak zobrazuje obrázek 6.2.

RPC volání se skládá ze čtyř částí:

1. odeslání požadavku,
2. přijetí a vyhodnocení požadavku ve vzdáleném gadgetu,
3. zaslání výsledku operace tazateli,
4. přijetí a vyhodnocení výsledku operace tazatelem.

[4]Flash Player Detection Kit lze získat na http://www.adobe.com/products/flashplayer/download/detection_kit/

Při odesílání požadavku je nutné specifikovat příjemce – identifikační číslo (ID) gadgetu, kterému má být volání doručeno. Příjemce musí toto volání obsloužit předem zaregistrovanou obslužnou funkcí. Výsledek jejího vykonání je následně odeslán zpět tazateli, který může mít pro přijetí této odpovědi taktéž zaregistrovanou obslužnou funkci.

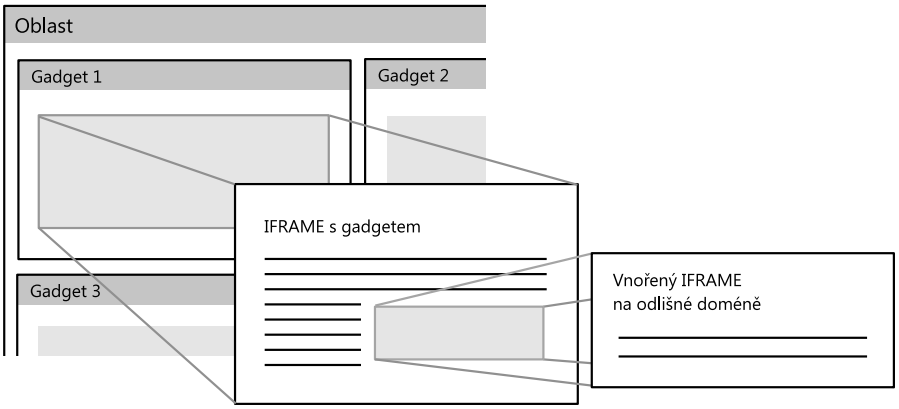
Způsoby volání

Při komunikaci mezi gadgety v rámci správce rozložení nedochází ze strany webového prohlížeče k žádným bezpečnostním omezením, protože všechny gadgety ve správci rozložení běží v rámci stejné domény. Lze tak provádět přímé vykonání javascriptové metody v jiném gadgetu (synchronně) a odpověď na RPC volání získat okamžitě.

V rámci rozšíření podpory pro RPC komunikaci i mezi gadgetem a v ním vnořeným plovoucím rámem (iframe) již však dochází k mezidoménové komunikaci, která je ve všech požadovaných prohlížečích z bezpečnostních důvodů zakázána, a přímá javascriptová volání vykonávat nelze. Proto je nutné pro tento případ implementovat speciální řešení.

Řešení mezidoménové komunikace

Při řešení mezidoménové komunikace bylo čerpáno především z [35] a [15].



Obrázek 6.2: Mezidoménová komunikace mezi gadgetem a vnořeným plovoucím rámem

Jak již bylo zmíněno, jsou-li tazatel a příjemce umístěni na různých doménách, není možné z bezpečnostních důvodů provádět přímé přístupy k vlastnostem jednotlivých dokumentů. Proto je nutné použít pro komunikaci jiné techniky, mezi které patří událost `message`, využití nechráněné vlastnosti `window.opener` a předání zprávy pomocí URL.

Událost `message` je nejnovější dostupnou technikou, která je součástí návrhu specifikace HTML5[5]. Využívá metody `window.postMessage`, která způsobí vygenerování události `message` pro zadané okno (`window`). Toto okno může událost zachytit, zpracovat a stejným způsobem zaslat odpověď [13]. Přestože se jedná teprve o součást návrhu budoucí specifikace HTML, podpora této techniky je v nejnovějších verzích webových prohlížečů velmi

[5] http://www.w3.org/TR/html5/comms.html#dom-window-postmessage-2

dobrá (Firefox od verze 3, Opera od verze 9, Internet Explorer od verze 8) [40]. Jedná se o asynchronní volání, jehož obsahem je textový řetězec. Proto bude nutné provést zakódování přenášených dat, a to nejenom dat odesílaných uživatelem, ale i dat řídících, která dané volání identifikují. Ideálním řešením je využít JSON, jehož podporu již máme v gadgets zabudovanou. Proto bude výhodné vytvořit dvě struktury – jednu pro volání typu dotaz a jednu pro volání typu odpověď – které budou obě informace obsahovat. Před odesláním požadavku pak stačí provést jejich serializaci.

Mezi prohlížeče, které událost `postMessage` nepodporují, patří i Internet Explorer verze 7 a nižší. Tyto verze Internet Exploreru nicméně poskytují jiný způsob, kdy lze pro komunikaci využít vlastnosti `window.opener`. Na tuto vlastnost se v uvedených verzích Internet Explorer bezpečnostní omezení nevztahují a lze přes ni v obou komunikujících oknech nastavit vzájemné reference. Přes takto nastavené reference lze následně provádět přímá volání, bez ohledu na rozdílnost domén.

Poslední technikou je způsob komunikace přes URL. Tato technika je nejméně spolehlivá, protože se spoléhá na předání dat jako součástí fragmentu URL adresy (část za znakem `#`). Plovoucí rám (iframe) změnu URL zaregistruje a podle jejího obsahu provede požadovanou operaci. Nevýhodou tohoto řešení je omezení délky URL, tedy i omezení množství odesílaných dat. Další nevýhodou je možnost pouze jednosměrné komunikace (z gadgetu do plovoucího rámu). Plovoucí rám by mohl gadget o výsledku informovat opět změnou svojí URL adresy, nicméně o takové změně není gadget nijak informován a musí ji aktivně kontrolovat v pravidelných intervalech. Lepším řešením je vytvoření speciální vrstvy (RPC relay), která bude vložena do plovoucího rámu a která bude běžet na stejné doméně jako gadget. Díky tomu, že gadget i RPC relay poběží na stejné doméně, mohou mezi sebou provádět přímá volání (přestože mezi nimi existuje ještě jedna vrstva s plovoucím rámem).

Výsledný RPC požadavek z gadgetu do plovoucího rámu pomocí metody URL pak vypadá následovně:

1. gadget změní URL adresu plovoucího rámu
2. plovoucí rám zaregistruje změnu URL a vyhodnotí požadavek
3. plovoucí rám vytvoří speciální vrstvu – RPC relay
4. v URL adrese RPC relay předá zakódovanou odpověď
5. RPC relay rozpozná, že její URL adresa byla změněna a požadavek předá gadgetu
6. gadget odpověď přijme a zpracuje

Otázkou zůstává, jak detekovat změnu URL. V úvahu přichází dva přístupy [34]:

1. aktivní (*polling*) a
2. pasivní (*non-polling*).

Při aktivním přístupu plovoucí rám sám v pravidelných intervalech kontroluje, zda ke změně URL nedošlo (porovnává současnou a předchozí hodnotu). Toto řešení je výpočetně náročnější a generuje nemalou režii. Pasivní přístup naproti tomu pracuje na principu obsluhy

událostí. Mezi takové události patří například událost `onresize`, která je vyvolána prohlížečem při změně velikosti daného okna. Z hlediska efektivnosti je tak daleko výhodnější pasivní přístup.

Při řešení identifikace změny URL lze pak využít právě zmíněnou změnu velikosti, kdy je po nastavení nové URL změněn rozměr plovoucího rámu. Tato změna vyvolá událost, kterou plovoucí rám zaregistruje a vyhodnotí jako nové RPC volání.

6.5.6 Skins – podpora přizpůsobení vzhledu

Modul skins umožňuje získat základní grafické schéma prostředí, ve kterém gadget běží. Jedná se o čtyři hodnoty (barva písma, pozadí a odkazů a adresa obrázku na pozadí), které by bylo možné načítat z konfiguračního souboru nebo získávat dynamicky za běhu správce. Nevýhodou využití konfiguračního souboru je nemožnost snadné změny vzhledu (skin) správce rozložení stránky. Nelze tak reagovat ani na případné budoucí uživatelské profily, kdy každý z uživatelů může využívat odlišný vzhled. Dynamické zjišťování na konfiguraci správce rozložení naopak závislé není, protože probíhá pomocí funkcí JavaScriptu. Navíc technická realizace dynamického zjišťování není nijak složitá (uvažované webové prohlížeče umožňují použitá CSS pravidla získat), a i proto je tento přístup lepší volbou.

6.5.7 Views – různé obsahy gadgetu v závislosti na módu zobrazení

Specifikace počítá s tím, že prostředí, ve kterém gadget běží, může umožňovat více formátů zobrazení – např. profilové, detailní či celoobrazovkové. Pro každé z těchto zobrazení je možné definovat odlišný obsah, nicméně není to podmínkou – není-li obsah pro dané zobrazení definován, je zobrazen obsah výchozí. Po konzultaci s českou částí týmu projektu KiWi bylo rozhodnuto, že ve správci rozložení má význam pouze zobrazení profilové a detailní.

Profilové zobrazení (označováno jako `HOME`) je zobrazením výchozím, kdy je viditelná celá pracovní plocha správce rozložení s jednotlivými komponentami.

Detailním zobrazením (označováno jako `CANVAS`) chápeme stav, kdy je vybraná komponenta umístěna do popředí v maximální možné velikosti a ostatní komponenty jsou umístěné na pozadí, případně nejsou viditelné. Přičemž platí, že v zobrazení `CANVAS` může být vždy maximálně jedna komponenta. Jedná se o zcela nový způsob zobrazení, který dosud není ve správci rozložení přítomen. A protože správce rozložení je klientskou aplikací, musí být umožněno měnit obě zobrazení přímo za běhu gadgetu. V případě změny pohledu z výchozího (`HOME`) na detailní (`CANVAS`) je možné uvažovat nad třemi řešeními:

1. Přesun existující komponenty na konec DOM správce rozložení a změna jejího vzhledu (velikosti a pozice) pomocí CSS.
   Toto řešení má výhodu v tom, že lze komponentu přesunout do libovolné, předem právě k tomuto účelu připravené, větve DOM, na kterou se aplikují speciální CSS pravidla. Nevýhodou je, že vyjmutí z DOM může ovlivnit i ostatní komponenty na stránce a eventuálně dokonce způsobit jejich pád. To se týká i editoru TinyMCE, který je na podobné operace velmi citlivý.

2. Vytvoření nové komponenty a její umístění na konec DOM.
   Princip podobný předchozímu návrhu, který ovšem nevyjímá existující komponentu, ale vytváří její duplikát, a tak ostatní komponenty neohrozí. Při duplikaci ovšem dochází pouze k přenosu obsahu, nikoliv stavu komponenty. Vždy tak dojde k jejímu znovunačtení (restartování) a původní stav je ztracen (včetně dat v paměti).

3. Posledním možným způsobem je do DOM správce rozložení vůbec nezasahovat a změny velikosti i pozice komponenty dosáhnout pomocí CSS.
   Tento přístup má nevýhodu v tom, že bude nutné sestavit sadu CSS pravidel, která budou mít vyšší prioritu než všechna dosud aplikovaná pravidla. Velkou výhodou ale je, že nemá-li gadget pro zobrazení `CANVAS` definovaný speciální obsah, je zachován jeho současný obsah i stav a nedochází k žádné ztrátě dat.

Pro náš účel je nejpříznivější poslední navrhnutý způsob – bez změny DOM. Dané řešení tak bude funkční nejen pro gadgety, ale i pro „obyčejné“ komponenty a bude vhodné ho začlenit jako novou část správce rozložení stránky.

Před změnou zobrazení může tvůrce gadgetu uložit obsah vybraných proměnných do paměti správce rozložení. Po změně zobrazení si pak o takto uložená data může zažádat a na jejich základě přizpůsobit stav gadgetu. Lze tak zachovávat souvislost mezi jednotlivými zobrazeními gadgetu.

6.5.8 Window – funkce pro práci s oknem gadgetu

Všechny metody z modulu window jsou ve správci rozložení již implementovány, nicméně bude nutné provést jejich úpravu. Metoda pro změnu výšky musí brát v úvahu skutečnou výšku obsahu v plovoucím rámci (`iframe`), nikoliv pouze výšku komponenty s gadgetem. I metodu pro změnu titulku komponenty je nutné z důvodu změny způsobu vkládání gadgetů (začlenění do `iframe`) implementovat zcela od začátku.

KAPITOLA 7

Implementace

Při implementaci byly provedeny některé úpravy v původních zdrojových souborech správce rozložení stránky (`KiWiLayoutManager.js`), bylo vytvořeno jeho rozšíření (`KiWiGadgetsLayoutManager.js`) a byla rozšířena speciální knihovna implementující Gadgets API (`KiWiGadgetsWrapper.js`). Pro lepší udržovatelnost byl také vytvořen konfigurační soubor (`KiWiGadgetsConfig.js`). Při implementaci byla použita konvence označení funkcí, jež jsou konstruktorem, velkým počátečním písmenem. Kód je zároveň dokumentovaný pomocí komentářů pro systém JsDoc Toolkit[1]. Poskytuje tedy další, podrobnější zdroj informací.

7.1 Úpravy správce rozložení stránky

Z kódu správce rozložení stránky byla odstraněna většina původní části týkající se podpory Gadgets API. Některé z metod byly přesunuty do samostatného modulu, jiné byly nahrazeny nebo zcela odstraněny.

7.1.1 Aktualizace externích knihoven

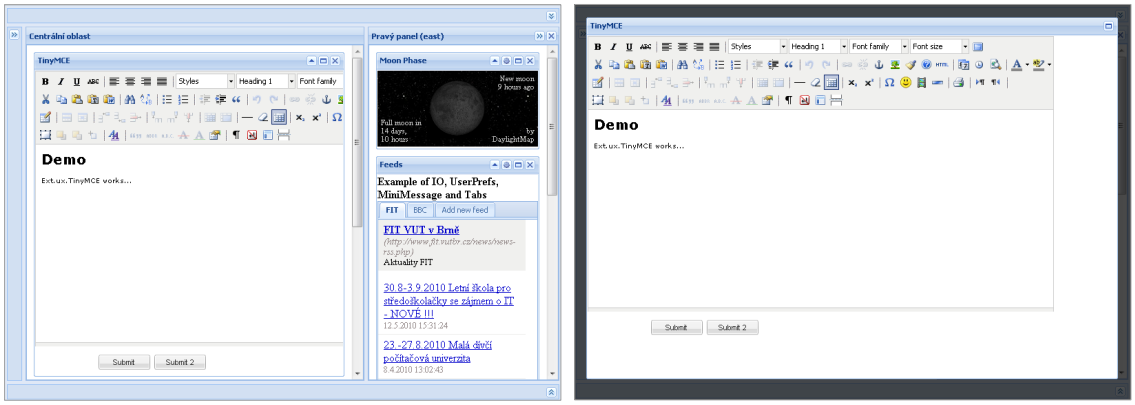
Před zahájením vlastní implementace bylo nutné provést aktualizaci knihovny Ext JS a TinyMCE na nejnovější verze. V případě Ext JS to znamenalo přechod z verze 2.1 na verzi 3.2.0, která zavedla mírně upravenou HTML strukturu komponent. V kódu správce rozložení tak bylo nutné provést úpravu přístupu k obsahu jednotlivých komponent. Aktualizace na novou verzi také způsobila nefunkčnost původního rozšíření (plugin) pro javascriptový editor TinyMCE. Autor tohoto rozšíření již ale vydal aktualizovanou verzi pro Ext JS řady 3[2], a bylo tak možné provést i jeho aktualizaci na verzi 0.7 beta1. Společně s novou verzí TinyMCE 3.2 se nakonec podařilo editor uvnitř správce rozložení stránky zprovoznit.

7.1.2 Podpora pro detailní zobrazení

Správce rozložení stránky byl rozšířen o podporu druhého způsobu zobrazení (kromě výchozího `HOME` i detailní `CANVAS`[3], obrázek 7.1), přičemž zobrazení lze měnit u všech komponent (nejen gadgetů). Protože ale běžné komponenty mohou mít obsah pouze pro jeden typ zobrazení (pro `HOME`), nemůže v jejich případě docházet při změně pohledu ke změně obsahu. Má-li běžná komponenta uvedeno více typů obsahu, je zobrazen obsah pro pohled `HOME` nebo, není-li uveden, poslední dostupný.

Změna pohledu je ve výchozím stavu povolena, nicméně lze ji u dané komponenty zakázat uvedením direktivy `<maximizing enabled="false"/>` v XML. Ke změně zobrazení (maximalizaci) byl zaveden nový nástroj umístěný v záhlaví dané komponenty. Po jeho aktivaci

[1] http://code.google.com/p/jsdoc-toolkit/
[2] http://www.extjs.com/forum/showthread.php?24787-Ext.ux.TinyMCE-TinyMCE-form-field-(v0.7b1)
[3] viz Views – různé obsahy gadgetu v závislosti na módu zobrazení str. 27



Obrázek 7.1: Zobrazení HOME (výchozí) vlevo, detailní zobrazení (CANVAS) vpravo

dojde k nalezení pozice dané komponenty (nalezení sloupce a oblasti, ve které leží) a nastavení připravených CSS tříd. Kromě nové pozice zajistí aplikování CSS pravidel i skrytí ostatních komponent.

Čeho pomocí CSS dosáhnout nelze, je nastavení nových rozměrů komponenty (aby její výška i šířka byla přes celou pracovní plochu), protože tyto hodnoty jsou přepočítávány Ext JS při každé změně rozvržení stránky. K tomuto účelu je nutné využít JavaScript, kterým je také nutné ošetřit `onresize` událost – dojde-li ke změně velikost okna prohlížeče, musí se komponenta v popředí těmto novým rozměrům přizpůsobit.

Aby byly ostatní prvky na stránce neaktivní, byla vytvořena podkladová vrstva (overlay). Ta je vygenerována před prvním přepnutím a je vložena na konec DOM správce rozložení. Tato vrstva má černou poloprůhlednou barvu a zamezuje propagaci událostí myši na ostatní komponenty. Při maximalizaci je podkladová vrstva zobrazena, při minimalizace zase skryta.

V souvislosti se změnou zobrazení byly zavedeny i čtyři nové události, které správce rozložení pomocí observeru komponentám rozesílá – `componentWillBeMaximized` (vygenerována před zahájením procesu maximalizace), `componentMaximized` (vygenerována po dokončení maximalizace), `componentWillBeMinimized` (vygenerována před zahájením minimalizace) a `componentMinimized` (vygenerována po dokončení minimalizace).

Veškeré informace, které si správce rozložení potřebuje v souvislosti se změnou zobrazení uschovat, jsou uložené ve struktuře `KiWiLayoutManager.fullscreen`.

7.1.3 Chybové zprávy

V souvislosti se zavedením nových funkcí vyvstala potřeba informovat uživatele o vzniklých chybách. Knihovna Ext JS sice obsahuje třídu `Ext.Msg.alert`, nicméně tato třída implementuje návrhový vzor jedináček (singleton), a není tedy možné zobrazit více chybových zpráv současně. Navíc každá nově zobrazená zpráva nahradí zprávu předchozí, a uživatel tudíž nemá přehled o historii vzniklých chyb.

Proto byla vytvořena metoda `KiWiLayoutManager.showError`, která zobrazuje chybovou zprávu formou modálního dialogového okna. Texty zpráv se načítají z lokalizovaných ja-

zykových souborů a pro lepší vypovídací hodnotu mohou obsahovat zástupné proměnné, které jsou před zobrazením zprávy nahrazeny za skutečné hodnoty. Chybová zpráva je tak srozumitelnější, protože může obsahovat přesný popis chyby, např. ve které komponentě chyba vznikla.

7.1.4 Generování identifikátorů komponent

Nemá-li v XML zdroji komponenta svůj identifikátor uveden, vygeneruje ho správce rozložení. Takto vygenerovaný identifikátor sestává z předpony (prefix) a pořadového čísla. U gadgetů ovšem musí být identifikátor celočíselný. Proto bylo nutné v jejich případě vytvořit samostatný generátor, `KiWiLayoutManagerGadgetsPlugin.getNextGadgetId`, který na základě pořadí komponenty vygeneruje číselnou hodnotu. Tu je ovšem nezbytné porovnat s ostatními identifikátory a ověřit, zda je v rámci XML skutečně unikátní. Není-li tomu tak, je nutné generování opakovat. Současně je nutné vygenerované identifikátory uschovávat (`KiWiLayoutManagerGadgetsPlugin.gadgetsGeneratedIds`), protože se předpokládá, že správce rozložení bude v budoucnu ukládat stav jednotlivých komponent na server. A při ukládání je nutné rozlišit, zda se jedná o identifikátor pevně daný (statický) nebo dynamicky vygenerovaný (který může být při příštím načtení odlišný).

7.1.5 Úprava načítání XML

Při načítání obsahu gadgetu z XML je třeba rozlišovat, o které zobrazení (pohled) se jedná. Načítání je navíc nutné opakovat, dokud jsou další pohledy k dispozici, a ukládat je do zvláštní struktury (pomocí `KiWiLayoutManagerGadgetsPlugin.registerGadgetContent`). Při dokončení načítání všech dostupných obsahů je pak nutné aktivovat obsah výchozí – pro pohled `HOME`.

7.1.6 Vyvolání události informující o dokončení načítání

Vyvolání události `layoutReady` je možné ve dvou situacích: ihned po načtení všech komponent, když není zapnuta podpora pro Google gadgets, nebo po načtení všech gadgetů, když podpora zapnuta je. V případě zapnuté podpory lze počet dosud nenačtených gadgetů ověřovat pomocí metody `KiWiLayoutManagerGadgetsPlugin.countUnloadedGadgets`. Tato metoda bere v úvahu všechny podmínky, za kterých lze gadget spustit, včetně načítání externích zpráv. Klesne-li počet nenačtených gadgetů na hodnotu 0, lze událost `layoutReady` vyvolat.

7.2 Rozšíření správce rozložení stránky

Pro rozšíření existujícího řešení byla vytvořena samostatná funkce `KiWiLayoutManagerGadgetsPlugin` umístěná v souboru `KiWiGadgetsLayoutManager.js`. Pro zjednodušení jejího názvu bude v následujícím textu využita zkrácená podoba `Plugin`. Tato funkce způsobí rozšíření existující instance objektu typu `KiWiLayoutManager` o nové vlastnosti nutné pro začlenění podpory gadgetů do správce rozložení. Funkci je nutné vyvolat v kontextu tohoto

existujícího objektu a před jejím vykonáním je nutné načíst konfiguraci gadgetů. Vlastní správce rozložení je nicméně funkční i bez její přítomnosti; je tak splněna podmínka pro implementaci formou nezávislého rozšíření.

7.2.1 Vložení nového gadgetu do správce rozložení

Načítání obsahu gadgetu probíhá podobně jako načítání ostatních komponent. Rozdíl nastává při zpracovávání dat. Gadget totiž může obsahovat obsah pro více zobrazení (v našem případě je důležitý obsah pro zobrazení `HOME` a `CANVAS`) a aby bylo možné mezi nimi přepínat, musí být uloženy v paměti. Ve všech obsazích je také nutné provést nahrazení proměnných a jazykových zpráv. Ukládání načteného obsahu do paměti se realizuje funkcí `Plugin.registerGadgetContent`. Uložením nového obsahu se současně gadgetu nastaví příznak, že má nový obsah a že je třeba vyčkat na jeho načtení. Tento příznak se využívá pro kontrolu stavu gadgetů v metodě `Plugin.countUnloadedGadgets` před vyvoláním `layoutReady` události.

Do správce rozložení stránky je gadget vložen formou plovoucího rámu (`iframe`), proto je nutné metodou `Plugin.createGadget` vygenerovat jeho HTML kód. Vygenerovaný HTML kód bude pro správce rozložení od této chvíle reprezentovat obsah dané komponenty. Daný element `<iframe>` má navíc speciální hodnotu atributu `id`, která je sestavena z předpony (prefix) a identifikátoru komponenty. Na základě jeho hodnoty pak dochází k identifikaci dané komponenty za běhu aplikace[4]. Tento plovoucí rám odkazuje na připravenou šablonu (`KiWiEmptyGadget.html`) a skutečným obsahem je naplněn až po její inicializaci.

Správce rozložení od této chvíle předpokládá, že pracuje s běžnou komponentou obsahující prvek `iframe`, a že se jedná o gadget, dále nerozlišuje.

7.2.2 Načtení šablony a spuštění gadgetu

Připravená šablona obsahuje ve své hlavičce (`head`) odkazy na javascriptové a CSS knihovny Ext JS a vytvořené knihovny pro podporu Gadgets API. Po načtení šablony je vyvolána metoda `Plugin.startGadget`, která na základě aktivního pohledu provede vložení odpovídajícího HTML obsahu do těla šablony. Protože při vložení HTML pomocí vlastnosti `innerHTML` elementu `body` nedojde k vykonání vložených elementů `<script>`, je nutné provést jejich aktivaci. Aktivaci vložených (inline) skriptů lze provést okamžitě, a to pouhým vyjmutím a znovu-vložením do odpovídající DOM větve (`KiWiLayoutManager.activateScripts`). U externích skriptů při tomto postupu ovšem dochází k asynchronnímu načítání, které negarantuje pořadí vykonání.

Proto byla navržena možnost synchronního načítání pomocí techniky AJAX[5]. Načtení externího skriptu pak probíhá přes proxy server[6] a jeho obsah je do dokumentu vložen přímo (inline). Nicméně je nutné si uvědomit, že synchronní načítání nemusí být vždy vyhovující (může způsobovat zamrznutí prohlížeče), proto lze způsob aktivace skriptů jednoduše nastavit v konfiguraci.

[4] Způsob identifikace komponent je popsán na straně 33.

[5] Přestože AJAX je navržen pro asynchronní požadavky, v odůvodněných případech jej lze využít i pro synchronní volání.

[6] Popis proxy serveru je uveden v Serverová část, strana 38.

Avšak vložení obsahu není jedinou činností, kterou je nutné vykonat. V případě, že gadget obsahuje jazykové zprávy, je nutné provést jejich načtení a dosazení. Zprávy jsou uložené v elementech `<Locale>`, přičemž je nutné respektovat jejich různou prioritu[7]. Při vyhodnocování jednotlivých elementů dochází nejdříve ke zpracování vložených (inline) zdrojů, tedy takových, kdy jsou hodnoty zpráv přítomny přímo v XML gadgetu. Poté dochází i k vyhodnocení externích zpráv, jejichž obsah je nejdříve nutné pomocí AJAX požadavků získat (`Plugin.loadGadgetsExternalLocale`).

Až když je obsah vložený a jazykové zprávy zpracované, lze gadgetu nastavit příznak, že je připraven k běhu. Pokud se jednalo o poslední gadget, který dosud načtený nebyl, je vyvolána událost `layoutReady` a tím je správce rozložení stránky oficiálně spuštěn.

7.2.3 Způsob identifikace komponent

Při vkládání nového gadgetu je součástí vygenerovaného HTML elementu `<iframe>` i atribut `id`, který obsahuje zakódovaný identifikátor dané komponenty[8]. Gadget může požádat o získání svého identifikátoru pomocí metody `Plugin.getMyGadgetsId`, které předá referenci na svůj dokument – vlastnost `window.document`. Správce rozložení pak následně projde všechny vytvořené gadgety a na základě porovnání předané reference s jednotlivými dokumenty nalezne odpovídající element `<iframe>` a z jeho atributu `id` hodnotu identifikátoru získá. Jednou získaná hodnota je uschována, a proto jsou opakované žádosti rychlejší.

Správce rozložení při identifikaci komponent také pracuje s metodami `Plugin.getGadgetsIframe` a `Plugin.getGadgetsIframeDocument`, které dle předaného identifikátoru vrací objekt `iframe`, resp. `document` daného gadgetu.

7.2.4 Kontrola podpory vlastností (features)

Byla zavedeno asociativní pole `Plugin.gadgetsFeatureParams`, které uchovává vlastnosti funkcí pro všechny gadgety. Toto pole je naplněno metodou `Plugin.loadGadgetsFeaturesParams`, která zároveň ověřuje, zda jsou všechny vyžadované (require) vlastnosti podporovány. Není-li tomu tak, zobrazí správce rozložení místo obsahu gadgetu chybovou zprávu informující uživatele, že daný gadget nemůže být zobrazen.

7.2.5 Konfigurace Gadgets

Pro snadnou konfiguraci implementovaného řešení byl vytvořen konfigurační objekt `gadgets.config` umístěný v souboru `KiWiGadgetsConfig.js`. Ten obsahuje adresu proxy serveru, adresu šablony prázdného gadgetu, seznam podporovaných vlastností a pohledů a další nastavení.

Součástí konfigurace je i hodnota nejnovější dostupné verze Adobe Flash Playeru. Detekce nainstalované verze totiž v Internet Exploreru vyžaduje sestupné ověřování (od nejnovější po nejstarší), jak je uvedeno v části Flash na straně 35.

[7] Jak je popsáno v Jazykové lokalizace na straně 20.
[8] Vložení nového gadgetu do správce rozložení je uvedeno na straně 32.

7.3 Gadgets API – jádro (Core API)

7.3.1 Prefs, JSON a Util

Existující implementace modulu Prefs byla upravena tak, aby využívala nový způsob identifikace komponent. Současně je pro formuláře s nastavením zavedena podpora pro více jazykových verzí.

Podpora pro JSON byla implementována podle návrhu – pomocí `Ext.util.JSON`.

Metoda `gadgets.getFeatureParameters` byla implementována s využitím dříve zmíněného asociativního pole `Plugin.gadgetsFeatureParams`. Metoda `gadgets.hasFeature` byla implementována s využitím konfiguračního souboru, který obsahuje seznam všech podporovaných vlastností.

7.3.2 IO

Pro komunikaci se vzdálenými doménami byl vytvořen jednoduchý proxy server v jazyce PHP[9]. Správce rozložení nově obsahuje metodu `Plugin.getGadgetsProxyUrl`, která na základě parametru vzdálené URL vrací HTTP adresu proxy serveru, který spojení zprostředkuje. Podporovány jsou HTTP metody GET, POST a HEAD.

Na základě typu požadavku bylo implementováno získávání dat z RSS a Atom zdrojů, a to z RSS 0.91 a 2.0 a Atom 1.0. Jak bylo zmíněno v návrhu, práce se jmennými prostory v XML, které mohou být využity u Atom zdrojů, není pomocí Ext JS možná, a proto nejsou jmenné prostory podporovány. Přestože atom zdroje mohou obsahovat různé formáty dat, podporovány jsou pouze typ `text` (prostý text) a `html`.

Vytvoření nového požadavku se realizuje pomocí statické metody `gadgets.io.makeRequest`, která provede analýzu předaných parametrů a na jejich základě vytvoří objekt typu `gadgets.io.RequestParams`. Následně je pomocí `Ext.Ajax.request` vytvořen AJAX požadavek na proxy server. Po přijetí odpovědi na tento požadavek je vytvořen objekt `gadgets.io.RequestResponse`, která na základě parametrů spojení (`RequestParams`) provede vyhodnocení odpovědi a zpracování dat. Zpracovaná data jsou nakonec předána uživatelem definované funkci (callback), čímž je požadavek dokončen.

Uživatelem definované funkci je předán objekt typu `gadgets.io.RequestResponse`, který obsahuje několik vlastností. Úplný text odpovědi serveru je k dispozici ve vlastnosti `RequestResponse.responseText`, hlavičky odpovědi v `headers`, návratový kód serveru v `rc`, originální `XMLHttpRequest` ve vlastnosti `xhrObject` a zpracovaná data ve vlastnosti `data`. Specifikace tuto strukturu přímo neurčuje [14], nicméně z příkladů implementace ji lze vyčíst[10].

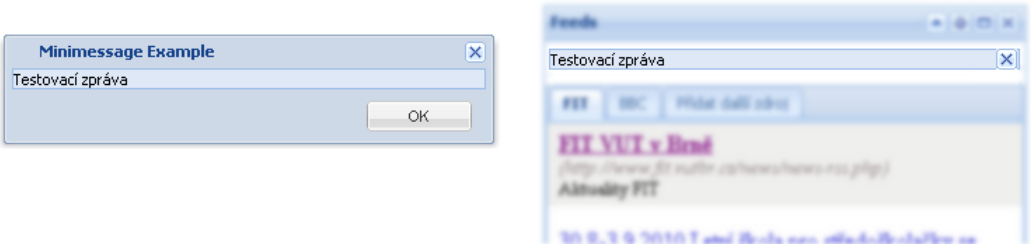
[9] Podrobnější popis proxy serveru je uveden v Serverová část, strana 38.

[10] http://code.google.com/intl/cs-CZ/apis/gadgets/docs/remote-content.html#Fetch_text

7.4 Gadgets API – Specifické moduly (Feature-Specific API)

7.4.1 MiniMessage

Modul MiniMessage byl implementován dvěma způsoby. Je-li při vytváření objektu (`new gadgets.MiniMessage`) uveden parametr s HTML kontejnerem, do kterého mají být zprávy vkládány, jsou generovány vložené (inline) zprávy. V opačném případě jsou zprávy zobrazovány do samostatného dialogového okna (obrázek 7.2).



Obrázek 7.2: Typy zpráv v modulu MiniMessage

7.4.2 TabSet, PubSub a Skins

Pro implementaci modulu TabSet byla využita komponenta TabPanel z Ext JS, jak bylo popsáno v kapitole Návrhu, části TabSet – správce záložek (tabs) a jejich obsahu. Metody `alignTabs` a `swapTabs` jsou vyvolatelné, ale jejich vykonání nemá žádný účinek.

Modul PubSub byl implementován s využitím observeru. Metoda `gadgets.pubsub.subscribe` provede zaregistrování objektu k příjmu zpráv daného typu, metoda `gadgets.pubsub.publish` pak zprávy publikuje. Aby mohl gadget zprávu publikovat, nemusí být přihlášen k jejímu odběru.

U modulu Skins byla implementována podpora dynamického zjišťování (za běhu aplikace) vzhledu správce rozložení. Hodnoty se načítají z těla prvního panelu (první komponenty) umístěného ve správci rozložení. Barva odkazu je získána dle barvy textu v záhlaví tohoto prvního panelu.

7.4.3 Flash

Pro detekci verzi Adobe Flash Playeru ve webovém prohlížeči byla využita metoda popsaná v Flash Player Detection Kit[11]. Pro vložení objektu do HTML byl využit element `object` v kombinaci s `embed` pro starší prohlížeče. Ukládání animací do cache (`gadgets.flash.embedCachedFlash`) není z důvodu využití pouze jednoduchého proxy serveru podporováno a metoda je pouze referencí na `gadgets.flash.embedFlash`.

V Internet Exploreru je Adobe Flash Player reprezentován ActiveX objektem. Každá verze Adobe Flash používá objekt s odlišným názvem (`ShockwaveFlash.ShockwaveFlash.X`, kde

[11] http://www.adobe.com/products/flashplayer/download/detection_kit/

X je číslo verze), a protože nás zajímá nejvyšší podporovaná verze, je nutné postupovat sestupně – od nejvyšší verze. Z tohoto důvodu je nejvyšší na trhu dostupná verze Flash Playeru uložena v konfiguraci.

7.4.4 RPC

Implementace RPC byla rozdělena do několika částí, dle zvolené metody. Byla vytvořena podpora pro přímé volání, pro událost `message`, využití vlastnosti `window.opener` v Internet Exploreru i přenos pomocí URL.

Před zahájením vlastního volání je zvolena jedna z uvedených metod. Nejdříve je proveden pokus o přímé volání, není-li z bezpečnostních důvodů dostupné, je zjištěno, zda prohlížeč podporuje událost `message`. Pokud událost podporována není a jedná se o Internet Explorer 6 nebo 7, je využit způsob pomocí `window.opener`. Ve všech ostatních případech se přikročí k variantě komunikace přes URL. Nutno podotknout, že v rámci komunikace mezi gadgety v rámci správce rozložení se vždy využije přímé volání. Ostatní metody přichází v úvahu až při komunikaci mezi gadgetem a v ním vloženým plovoucím rámem (iframe), což je rozšíření nad rámec původní specifikace.

Hledání příjemce volání probíhá takto:

1. Je-li jako příjemce uvedena hodnota ".." (dvě tečky) nebo prázdný řetězec, je cílem volání rodičovské okno.
2. Existuje-li ve správci rozložení stránky komponenta, jejíž identifikační kód odpovídá předané hodnotě, je cílem volání ona.
3. Existuje-li v gadgetu plovoucí rám (iframe), jehož atribut `id` odpovídá předané hodnotě, je cílem volání on.
4. Cíl neexistuje, volání je zahozeno.

Příjemce volání vyhodnotí a vrátí výsledek operace. V případě přímého volání a využití `window.opener` je tento proces synchronní a dochází k okamžitému vykonání. U ostatních metod se jedná o asynchronní volání a jeho výsledek je odesílateli navrácen opět formou RPC volání.

Komunikace s vloženým plovoucím rámem

Jako rozšíření podpory RPC byla implementována i možnost, kdy gadget komunikuje s dalším, v ním vloženým plovoucím rámem. Nevýhodou je, že aby byla komunikace možná, musí vložený plovoucí rám připojit knihovnu `KiWiGadgetsRPC.js`, `KiWiGadgetsWrapper.js` i javascriptové knihovny Ext JS. Odstranění nebo minimalizaci těchto závislostí lze uvažovat jako jedno z vylepšení do případné budoucí verze.

Je-li pro komunikaci s tímto rámem využita metoda události `message` nebo URL, je nutné zprávu požadavku i odpovědi zakódovat, protože budou přenášeny ve formě řetězce (string). Proto byly vytvořeny objekty `gadgets.rpc.RequestMsg` a `gadgets.rpc.ResponseMsg`, které obsahují kromě vlastních dat i data řídící.

Při komunikaci přes URL byla vytvořena speciální vrstva (RPC relay), která je umístěna v souboru `KiWiGadgetsRPCRelay.html` a zajišťuje volání gadgetu z vnořeného plovoucího rámu (iframe). Tato vrstva je vložena do plovoucího rámu opět formou plovoucího rámu, je v něm absolutně pozicována a skryta za levý okraj okna, takže není viditelná. Vrstva, stejně jako vnořený plovoucí rám, využívá pasivní informování o změně URL – reakci na `onresize` událost. Zde bylo nutné provést rozlišení podle prohlížeče, protože u malých rozměrů okna negeneruje Internet Explorer `onresize` událost při změně výšky. Z pohledu uživatele je ale změna výšky výhodnější, než změna šířky, protože méně ovlivňuje rozvržení (layout) stránky. Proto u Internet Exploreru probíhá změna šířky a u ostatních prohlížečů změna výšky.

Chceme-li v případě Internet Exploreru 6 a 7 využít vlastnosti `window.opener`, je nutné nejdříve nastavit vzájemné reference[12]. Tyto reference lze nastavit pomocí statické metody `gadgets.rpc.setupFrame`, která jako parametr očekává plovoucí rám, s nímž má být vazba nastavena. Tato metoda musí být vykonána před načtením daného plovoucího rámu, protože opačná vazba je nastavena právě při jeho načtení (`onload`).

7.4.5 Views

Podpora pro změnu pohledu byla začleněna přímo do správce rozložení stránky[13]. Změna pohledu gadgetu se realizuje pomocí metody `gadgets.views.requestNavigateTo`, která s použitím metody správce rozložení `Plugin.switchGadgetsView` vyvolá změnu obsahu i změnu zobrazení. Ke změně obsahu dochází pouze tehdy, má-li gadget pro nový pohled obsah definovaný. V opačném případě je zachován obsah současný.

Změna obsahu gadgetu probíhá znovunačtením šablony. Správce rozložení si poznamená, že pohled byl změněn, a šablonu obnoví. Šablona po svém obnovení vyvolá metodu `gadgets.startGadget`[14], a protože správce rozložení vrací obsah pro aktuální pohled (který již byl změněn), je šablona naplněna novým obsahem. V tom případě je pouze nutné ošetřit vyvolání událostí zaregistrovaných pomocí `gadgets.util.registerOnLoadHandler` a vykonat je okamžitě, protože událost `layoutReady` již byla vygenerována dříve, a zaregistrované funkce by se tak jinak vůbec nevykonaly.

Součástí podpory modulu views je i dosazení proměnných do URL šablony dle návrhu URI Template[15]. Pro tuto funkčnost byla využita externí knihovna *URI Template Library* od James M. Snell[16]. Knihovna je umístěna v souboru `KiWiGadgetsURITemplates.js`.

7.4.6 Window

Funkce pro zjištění výšky obsahu gadgetu nyní pracuje s hodnotami `offsetHeight` a `scrollHeight`, které obsahují skutečnou výšku obsahu bez ohledu na jeho zobrazenou část.

Změna titulku pomocí `gadgets.window.setTitle` byla přizpůsobena na novou identifikaci komponent, kdy gadget nejdříve pomocí `Plugin.getMyGadgetsId` získá svůj identifikátor, a pak nastaví titulek odpovídající komponentě.

[12] Řešení mezidoménové komunikace strana 25
[13] Podpora pro detailní zobrazení, strana 29
[14] Opakuje se proces načítání, který byl popsán v Načtení šablony a spuštění gadgetu na straně 32.
[15] http://bitworking.org/projects/URI-Templates/spec/draft-gregorio-uritemplate-03.html
[16] http://www.ibm.com/developerworks/web/library/wa-uri/index.html

7.5 Legacy API

Dle *Gadgets.* Migration Guide*[17] byla vytvořena sada funkcí poskytujících podporu pro předchozí verzi API (nyní zastaralou). Zdrojové kódy převodních funkcí byly převzaty přímo z uvedeného článku; většinou se ale jedná pouze o nastavení mapování starého názvu funkce na nový. Tyto funkce jsou umístěny na konci souboru `KiWiGadgetsWrapper.js`, a lze je tak v budoucnu jednoduše odstranit.

7.6 Serverová část

Na straně serveru byl implementován pouze jeden skript – jednoduchý proxy server – `KiWiGadgetsProxy.php`. Tento skript je vytvořen v programovacím jazyce PHP a obsahuje obecnou třídu pojmenovanou `Proxy`. Tato třída zajišťuje přijmutí požadavku a jeho předání na zadanou vzdálenou URL (včetně přijatých HTTP hlaviček, až na některé výjimky). Odpověď, kterou od serveru obdrží, pak v téměř nezměněné podobě (až na vybrané HTTP hlavičky) vrátí tazateli. Proxy server podporuje metody GET, POST a HEAD. Pro otevření spojení na vzdálený server využívá PHP funkcí `stream_context_create` a `file_get_contents`.

Kromě vlastní třídy `Proxy` obsahuje soubor i kód pro vytvoření jedné její instance a získání dat z požadované vzdálené URL. Vzdálená URL adresa a požadovaná HTTP metoda jsou proxy serveru předávány v parametrech URL.

7.7 Neimplementované metody

Některé metody, které nejsou pro funkčnost gadgetů nezbytné, nebyly v rámci této práce implementovány. Jejich deklarace nicméně součástí knihoven jsou, takže jejich volání chybu nezpůsobí, pouze se nijak neprojeví. Mezi tyto metody patří zejména metody týkající se platformy OpenSocial (především součástí `gadgets.io`). Také nejsou podporovány metody `gadgets.TabSet.alignTabs` a `gadgets.TabSet.swapTabs`, protože použitá komponenta `Ext.TabPanel` tuto funkcionalitu nepodporuje. Z důvodu jednoduchosti proxy serveru není implementována vyrovnávací paměť, takže metody, které data z cache načítají, jsou ve skutečnosti pouze reference na metody, které vyrovnávací paměť nevyužívají.

Dále nejsou implementovány některé z modulů, které sice jsou součástí specifikace, ale v implementaci společností Google nejsou zmíněny (`gadgets.log` a `gadgets.i18n`).

[17] http://code.google.com/intl/cs-CZ/apis/gadgets/docs/migration.html

KAPITOLA 8

Testování

8.1 Testovací gadgety

Již během implementace bylo prováděno ověřování funkčnosti pomocí vybraných gadgetů získaných z adresáře společnosti Google[1]. Současně byla také testována funkčnost ukázkových kódů, které jsou součástí *gadgets.* API Developer's Guide*[2].

Pro demonstraci funkčnosti byla vytvořena sada pěti ukázkových gadgetů, které využívají metod všech implementovaných modulů a reprezentují plnohodnotné aplikace. Funkčnost těchto gadgetů byla následně uživateli otestována ve všech vyžadovaných prohlížečích[3] a všechny nalezené nedostatky byly do konečné verze opraveny.

8.2 Jednotkové (unit) testy

Pro ověření funkčnosti vzhledem ke specifikaci Gadgets API (funkcionální testování, black box metoda [33]) byla pomocí knihovny `Ext.ux.UnitTest`[4] vytvořena sada jednotkových (unit) testů. Tato sada byla určena pro testování vybraných metod s předem definovanými vstupy a očekávanými výstupy.

Komponenty uživatelského rozhraní tímto způsobem testovat nelze, v jejich případě proto bylo provedeno uživatelské testování pomocí zmíněných testovacích gadgetů.

8.3 Profilování a porovnání výkonu

Měření času, který je k vykreslení stránky potřeba, není jednoduché. Použití profilovacích nástrojů ve webových prohlížečích přináší poměrně velkou režii, která celé měření zkresluje. Nástroje navíc neumožňují detekovat čas dosažení určité události, ale zobrazují pouze přehled volání jednotlivých funkcí. Měření proto probíhalo pomocí JavaScriptu a objektu `Date`, konkrétně jeho metody `getTime`, která vrací počet milisekund od 1. 1. 1970. První čas byl změřen ihned na začátku dokumentu (v části `head`, před načítáním všech knihoven) a druhý po vyvolání události `layoutReady`. Měření probíhalo na lokálním počítači s tím, že i veškeré načítané soubory byly umístěny na lokálním disku. Spuštění měření probíhalo navštívením dané stránky (nikoliv klávesou `F5`), a to proto, aby prohlížeče co nejvíce dat načítaly ze své vyrovnávací paměti (ve všech případech byla cache v prohlížeči povolena).

V každém prohlížeči bylo měření opakováno 10krát a rozdíly koncových a počátečních časů zprůměrovány. Zaokrouhlené vypočtené průměrné hodnoty jsou uvedeny v tabulce 8.1. Nejedná se samozřejmě o čísla absolutní, spíše je nutné všímat si rozdílů mezi jednotlivými prohlížeči a počtem gadgetů na stránce.

[1] http://www.google.com/ig/directory
[2] http://code.google.com/intl/cs-CZ/apis/gadgets/docs/dev_guide.html
[3] Požadované podporované prohlížeče jsou uvedeny na straně 4 v části Požadovaná programová podpora.
[4] http://code.google.com/p/extjs-unit-test/

	Chrome 4.1.249	**Opera** 10.53	**Firefox** 3.6.3	**IE** 8
Původní verze	831 ms	818 ms	1 113 ms	867 ms
Aktualizace Ext JS 3.0	1 109 ms	1 055 ms	1 377 ms	1 188 ms
Upravená verze bez gadgetu	1 075 ms	1 129 ms	1 450 ms	1 269 ms
Upravená verze s jedním gadgetem	1 712 ms	1 354 ms	1 865 ms	1 431 ms
Upravená verze se třemi gadgety	2 432 ms	1 582 ms	2 390 ms	1 603 ms
Upravená verze s pěti gadgety	3 023 ms	1 842 ms	3 011 ms	1 895 ms

Tabulka 8.1: Porovnání dob načítání v prohlížečích

Z tabulky je patrné, že aktualizovaná verze systému běží pomaleji než verze původní. Toto zpomalení, které je patrné i z pohledu uživatele, nastalo již pouhou aktualizací knihovny Ext JS na verzi 3.2. Vlastní upravení a rozšíření správce rozložení žádné další významné zpomalení nezpůsobilo. Poměrně velké rozdíly při načítání stránky ovšem způsobuje přítomnost gadgetů, které jsou vložené ve formě plovoucích rámců (iframe). Prohlížeč totiž v takové situaci vykresluje více dokumentů současně, s čímž si zdaleka ne všechny prohlížeče dokáží poradit. Nejvýrazněji je tato skutečnost patrná u prohlížeče Google Chrome, který v původním správci rozložení patřil k nejrychlejším, ale u více gadgetů se řadí k nejpomalejším.

KAPITOLA 9

Závěr

Po poměrně detailní analýze správce rozložení stránky projektu KiWi i platformy Google Gadgets bylo navrženo řešení, které umožnilo nejen začlenit podporu pro Google Gadgets, ale i rozšířit funkčnost vlastního správce rozložení. Jedná se svým způsobem o řešení neobvyklé, protože téměř veškerá funkcionalita je implementována formou JavaScriptu na straně klienta. Podporovány jsou všechny podstatné moduly Gadgets API, proto by vytvořené rozšíření mělo nalézt uplatnění v praxi. O jeho dostatečné kvalitě snad svědčí i to, že bylo českou částí týmu projektu KiWi odzkoušeno a splnilo akceptační testy pro zařazení do projektu.

Rozšíření bylo otestováno formou uživatelského testování a jednotkových (unit) testů, nicméně vzhledem k poměrně rozsáhlé množině možných vstupů je možné, že se nové chyby objeví během delšího provozu. Vzhledem k modulovému uspořádání a objektové koncepci by ale jejich oprava neměla činit problémy. Stejně tak je možné rozšířit podporu o další, nové moduly, které v této fázi implementovány nebyly.

V dalších fázích vývoje by bylo vhodné zaměřit se na optimalizaci rychlosti načítání stránky v případě, kdy obsahuje více gadgetů, protože současné řešení je při větším počtu gadgetů poměrně pomalé. Další možná rozšíření se týkají přímo správce rozložení stránky, kdy by bylo možné implementovat ukládání stavu pracovní plochy na server, uživatelské vzhledy (skin) nebo snadnější vkládání nových komponent.

Dle dostupných informací se na rozvoji správce rozložení stránky projektu KiWi podílí i další vývojáři, proto lze očekávat, že se do budoucna kromě Google Gadgets rozšíří podpora i na další podobné platformy.

Literatura

[1] *Document Object Model (DOM) Level 3 Core Specification: Version 1.0* [online]. 07 April 2004 [cit. 2010-04-28]. Dostupné z: <http://www.w3.org/TR/DOM-Level-3-Core>

[2] *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification* [online]. 08 September 2009 [cit. 2010-04-28]. Dostupné z: <http://www.w3.org/TR/CSS2/>

[3] *Mozilla Developer Center – JavaScript* [online]. 2010, last modified 03 Apr 2010 [cit. 2010-04-28]. Dostupné z: <https://developer.mozilla.org/en/JavaScript>

[4] *Mozilla Developer Center – DOM Levels* [online]. 2010, last modified 08 Feb 2010 [cit. 2010-04-28]. Dostupné z: <https://developer.mozilla.org/En/DOM_Levels>

[5] *Mozilla Developer Center – DHTML* [online]. 2010, last modified 17 Feb 2010 [cit. 2010-04-28]. Dostupné z: <https://developer.mozilla.org/cs/DHTML>

[6] *FAQ – WHATWG Wiki* [online]. 2010, last modified on 26 April 2010 [cit. 2010-04-28]. Dostupné z: <http://wiki.whatwg.org/wiki/FAQ#When_will_HTML5_be_finished.3F>

[7] *Implementations in Web browsers – WHATWG Wiki* [online]. 2010, last modified on 31 March 2010 [cit. 2010-04-28]. Dostupné z: <http://wiki.whatwg.org/wiki/Implementations_in_Web_browsers>

[8] *PHP: History of PHP* [online]. 2010, last updated Fri, 30 Apr 2010 [cit. 2010-05-05]. Dostupné z: <http://www.php.net/manual/en/history.php.php>

[9] *PHP: Hypertext Preprocessor* [online]. 2010, last updated Wed May 05 2010 [cit. 2010-05-05]. Dostupné z: <http://www.php.net>

[10] *HTML 4.01 Specification* [online]. 24 December 1999 [cit. 2010-04-28]. Dostupné z: <http://www.w3.org/TR/html401/>

[11] *XHTML™1.0 The Extensible HyperText Markup Language (Second Edition): A Reformulation of HTML 4 in XML 1.0* [online]. 26 January 2000, revised 1 August 2002 [cit. 2010-04-28]. Dostupné z: <http://www.w3.org/TR/xhtml1/>

[12] *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. 26 November 2008 [cit. 2010-04-28]. Dostupné z: <http://www.w3.org/TR/REC-xml/>

[13] *HTML 5: A vocabulary and associated APIs for HTML and XHTML* [online]. 4 March 2010 [cit. 2010-04-28]. Dostupné z: <http://www.w3.org/TR/html5/>

[14] *OpenSocial Gadgets API Specification v0.9* [online]. April 15, 2009 [cit. 2010-04-28]. Dostupné z: <http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/Gadgets-API-Specification.html>

[15] *Apache Shindig* [online]. c2010 [cit. 2009-12-22]. Dostupné z: <http://shindig.apache.org/>

[16] *Ext JS – JavaScript Framework and RIA Platform* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://www.extjs.com>

[17] *Gadgets and Internationalization (i18n) – Gadgets API – Google Code* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://code.google.com/intl/cs-CZ/apis/gadgets/docs/i18n.html>

[18] *gadgets.* API Developer's Guide - Gadgets API - Google Code* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://code.google.com/intl/cs-CZ/apis/gadgets/docs/dev_guide.html>

[19] *Gadgets API – Google Code* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://code.google.com/intl/cs-CZ/apis/gadgets/>

[20] *Gadgets.* Migration Guide - Gadgets API - Google Code* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://code.google.com/intl/cs-CZ/apis/gadgets/docs/migration.html>

[21] *Gadgets XML Reference* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://code.google.com/intl/cs-CZ/apis/gadgets/docs/reference.html>

[22] *Mozilla.org – JavaScript 1.5* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://www.mozilla.org/js/js15.html>

[23] *OpenSocial – Google Code* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://code.google.com/intl/cs-CZ/apis/opensocial/>

[24] *Rich Internet Application* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://www.symbio.cz/slovnik/rich-internet-application.html>

[25] *Sun Trademarks* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://www.sun.com/suntrademarks/>

[26] Crockford, D.: *JavaScript: The Good Parts*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2008, ISBN 978-0-596-51774-8.

[27] Dytrych, J.: *Vývoj webových komponent AJAX v prostředí ExtJS*. Diplomová práce, FIT VUT v Brně, 2009.

[28] Garrett, J. J.: *Ajax: A New Approach to Web Applications* [online]. February 18, 2005 [cit. 2010-04-28]. Dostupné z: <http://www.adaptivepath.com/ideas/essays/archives/000385.php>

[29] Hammond, D.: *Web browser CSS support* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://www.webdevout.net/browser-support-css>

[30] Hammond, D.: *Web browser DOM support* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://www.webdevout.net/browser-support-dom>

[31] Hammond, D.: *Web browser standards support summary* [online]. c2010 [cit. 2010-04-28]. Dostupné z: <http://www.webdevout.net/browser-support-summary>

[32] Johansson, R.: *Dump iframes and use object elements instead* [online]. December 14, 2006, update January 2010 [cit. 2010-04-28]. Dostupné z: <http://www.456bereastreet.com/archive/200612/dump_iframes_and_use_object_elements_instead/>

[33] Křena, B.; Kočí, R.: Úvod do softwarového inženýrství, 2006, studijní opora pro předmět IUS, FIT VUT, Brno.

[34] Lawson, P.: *Another Cross Domain iFrame Communication Technique* [online]. August 17, 2009 [cit. 2009-04-28]. Dostupné z: <http://shouldersofgiants.co.uk/Blog/post/2009/08/17/Another-Cross-Domain-iFrame-Communication-Technique.aspx>

[35] Mahemoff, M.: *Cross-Domain Communication with IFrames* [online]. March 31st, 2008 [cit. 2010-04-28]. Dostupné z: <http://softwareas.com/cross-domain-communication-with-iframes>

[36] Odell, D.: *Pro JavaScript RIA Techniques: Best Practices, Performance, and Presentation*. [Berkeley, CA, USA]: Apress, Inc., 2009, ISBN 978-1-4302-1934-7.

[37] Pecinovský, R.: *Návrhové vzory: 33 vzorových postupů pro objektové programování*. Brno: Computer Press, 2007, ISBN 978-80-251-1582-4.

[38] Resig, J.: *JavaScript a Ajax: Moderní programování webových aplikací*. Brno: Computer Press, 2007, ISBN 978-80-251-1824-5.

[39] Resig, J.: *Versions of JavaScript* [online]. April 22nd, 2008 [cit. 2010-04-28]. Dostupné z: <http://ejohn.org/blog/versions-of-javascript/>

[40] Resig, J.: *postMessage API Changes* [online]. May 22nd, 2008 [cit. 2010-04-28]. Dostupné z: <http://ejohn.org/blog/postmessage-api-changes/>

[41] Vrána, J.: *Verze PHP v ČR – únor 2010* [online]. 5.2.2010 [cit. 2010-04-28]. Dostupné z: <http://php.vrana.cz/verze-php-v-cr-unor-2010.php>

[42] Zakas, N. C.; McPeak, J.; Fawcett, J.: *Ajax: Profesionálně*. Brno: Zoner Press, 2007, ISBN 978-80-86815-77-0.

PŘÍLOHY

Seznam příloh

PŘÍLOHA A

Dokumentace pro vývojáře vytvářející komponenty

Text této dokumentace je se svolením autora převzat z [27] a aktualizován tak, aby reflektoval provedené změny.

Tato dokumentace je určena pro vývojáře komponentů a obsahuje detailní popis API správce rozložení. Obsahuje též popis struktury konfiguračního XML včetně detailního popisu významu jednotlivých elementů.

A.1 Správce rozložení

Konstruktor této třídy pouze vytvoří vlastnosti objektu a přiřadí jim výchozí hodnoty. Veškerá činnost je prováděna voláním metod. Objekt má vlastnosti s řetězci určené pro lokalizaci (viz dále) a následující vlastnosti:

- `loadPageFrom` – adresa, ze které se načítá XML s obsahem stránky. Tato veřejná vlastnost musí být nastavena před voláním funkce `loadPage()` (jinak je uskutečněn pokus o načtení z example.com)
- `gadgetsSupport` – vlastnost, která umožňuje zapnout podporu pro komponenty typu Google Gadgets. Výchozí hodnota je `false` (podpora Google Gadgets vypnuta). V případě zapnuté podpory Google Gadgets je nutné připojit knihovny `KiWiGadgetsConfig.js` a `KiWiGadgetsLayoutManager.js`.
- `gadgetsConfig` – konfigurační objekt pro Google Gadgets. Je-li podpora Google Gadgets zapnuta, je nutné ho inicializovat na `gadgets.config`.
- `showLoadingProgress` – vlastnost umožňující deaktivaci progressbaru zobrazeného při načítání konfiguračního XML. Výchozí hodnota je `true` (progressbar bude zobrazen). V případě nastavení na `false` progressbar nebude zobrazen a element pro jeho umístění nemusí být uveden v indexové stránce.
- `viewport` – reference na objekt viewportu (obsahuje veškeré objekty tvořící stránku). Komponenty by ji neměly běžně využívat, ale může být využita při ladění.

Správce rozložení má také následující metody:

- `loadPage()`
  - metoda pro načtení stránky, provádí následující činnost:
    - Zobrazí progressbar do elementu <`div id="layoutLoadingProgress"/`> (pouze pokud není jeho zobrazení zakázáno). Tento progressbar bude animován maximálně po dobu 60s a následně animace skončí, aby bylo signalizováno podezřele dlouhé načítání.
    - Načte ze serveru XML s obsahem stránky.

    * Zobrazí dialogové okno s progressbarem, aby uživatel nemohl zasahovat do neinicializované stránky v průběhu jejího sestavování.
    * Analyzuje XML a sestaví konfigurační objekt pro viewport. Pokud XML není validní, dojde v některých prohlížečích k chybě či zamrznutí (nelze vhodně a efektivně ošetřit ve všech prohlížečích). Pokud v XML chybí nějaké komponenty, jsou nahrazeny výplňovými (při vytvoření viewportu musí být v každé vytvořené oblasti v každém sloupci nejméně 1 komponent a současně 1. komponent ve sloupci musí být v nultém řádku a mezi řádky nesmí být mezery). Při analýze XML sestaví také některé stavové objekty v observeru, které obsahují informace o rozložení stránky, komponentech a jejich nastavení.
    * Vytvoří viewport.
    * Odstraní výplňové komponenty.
    * Provede dokončení inicializace formulářů s nastaveními komponentů.
    * Spustí skripty v komponentech tak, že každý uzel DOM stromu se značkou <`script`> je nahrazen nově vytvořeným shodným uzlem (obsah značek `script` není analyzován automaticky, protože uzly se skripty nejsou do DOM stromu vloženy spravným způsobem).
    * Přiřadí oblastem funkce pro obsluhu událostí.
    * Skryje dialogové okno, progressbar a element <`div id="layoutLoading"`>
    * V případě, kdy je vypnuta podpora Google Gadgets nebo jsou již všechny gadgety načtené, vloží do stránky skript, který vyvolá událost dokončení inicializace stránky („`layoutReady`“), kterou mohou komponenty využít jako náhradu za metodu `Ext.onReady()`. Nejsou-li všechny gadgety načtené, je tento skript vložený později.

- `getComponent(componentId)`
  - Popis: Získá referenci na komponent s daným id. Každý komponent je objekt třídy `Ext.panel,` a pro manipulaci s komponentem lze tedy využít většinu možností této třídy (viz dále).
  - Parametry:
    * `componentId` – id komponentu, na nějž se má získat reference
  - Vrací referenci na komponent
- `getComponentBody(componentId)`
  - Popis: Získá referenci na tělo komponentu. Komponenty, které mají formulář s nastavením, mají tělo ve vnořeném panelu pro obsah, komponenty bez nastavení jej mají přímo v panelu komponentu. Tato metoda poskytuje jednotné rozhraní pro přístup k tělu komponentu.
  - Parametry:
    * `componentId` – id komponentu, na jehož tělo se má získat reference
  - Vrací referenci na tělo komponentu

- `getComponentLocation(componentId)`
  - Popis: Získá objekt s informacemi o pozici komponentu.
  - Parametry:
    - `componentId` – id komponentu, jehož pozice se má získat
  - Vrací objekt s následujícími vlastnostmi:
    - `area` – oblast, ve které se komponent nachází (north, east, south, west nebo center)
    - `col` – sloupec, ve kterém se komponent nachází (číslovány od 0)
    - `row` – řádek, ve kterém se komponent nachází (číslovány od 0)
- `getComponentPreferences(componentId)`
  - Popis: Metoda pro získání reference na pole s uživatelskými nastaveními komponentu (`UserPref`). Získanou referenci nelze využít ke změně nastavení komponentu!
  - Parametry:
    - `componentId` – id komponentu, jehož uživatelská nastavení se mají získat
  - Vrací referenci na pole objektů s uživatelskými nastaveními komponentu indexované názvy položek nastavení. Každý objekt v tomto poli má nasledující vlastnosti:
    - `dataType` – typ pole ('string', 'bool', 'enum', 'list' nebo 'hidden')
    - `displayName` – zobrazovaný název pole
    - `required` – určuje, zda je pole povinné
    - `value` – hodnota
    - `enumValues` – pouze u typu enum, obsahuje položky výběru v poli objektů s následujícími vlastnostmi:
      - `displayValue` – zobrazovaná položka výběru
      - `value` – hodnota položky výběru
- `getComponentPrefsItem(componentId, prefName)`
  - Popis: Metoda pro získání položky uživatelského nastavení komponentu.
  - Parametry:
    - `componentId` – id komponentu, jehož položka nastavení se má získat
    - `prefName` – Název položky nastavení, která se má získat
  - Vrací požadovanou položku nastavení jako objekt s následujícími vlastnostmi:
    - `dataType` – typ pole (`'string'`, `'bool'`, `'enum'`, `'list'` nebo `'hidden'`)
    - `displayName` – zobrazovaný název pole
    - `required` – určuje, zda je pole povinné
    - `value` – hodnota
    - `enumValues` – pouze u typu enum, obsahuje položky výběru v poli objektů s následujícími vlastnostmi:
      - `displayValue` – zobrazovaná položka výběru
      - `value` – hodnota položky výběru

- `setComponentPrefsItem(componentId, prefName, newValue)`
  - Popis: Metoda pro nastavení hodnoty položky uživatelských nastavení komponentu (`UserPref`).
  - Parametry:
    * `componentId` – id komponentu, jehož nastavení se má měnit
    * `prefName` – název položky nastavení
    * `newValue` – nová hodnota položky nastavení
  - Vyvolá událost změny uživatelských nastavení komponentu (popis viz níže).
- `refresh()`
  - Popis: Metoda pro obnovení stránky.
    Pouze zavolá `window.location.reload()`
- `getAreasInfo()`
  - Popis: Metoda pro získání reference na objekt s informacemi o oblastech stránky. Referencovaný objekt není dovoleno měnit.
  - Vrací referenci na objekt s informacemi o oblastech, který má následující veřejné vlastnosti:
    * `center` – objekt s informacemi o centrální oblasti, který má následující veřejné vlastnosti:
      · `presented` – udává, zda je oblast ve stránce obsažena
      · `collapsible` – udává, zda je dovoleno skrývání těla oblasti
      · `collapsed` – udává, zda je tělo oblasti skryto
      · `cols` – pole informací o sloupcích, každa položka v tomto poli je tvořena objektem s vlastností `colWidth` udávající šířku sloupce
      · `title` – titulek oblasti – tuto vlastnost má jen když je oblast ve stránce obsažena (`presented == true`, u centrální vždy splněno).
    * `north` – objekt s informacemi o horní oblasti. Má stejné vlastnosti jako objekt pro centrální oblast, a pokud je oblast ve stránce obsažena, má navíc tyto vlastnosti:
      · `defaultSize` – výchozí výška oblasti
      · `minSize` – minimální výška oblasti při změně velikosti
      · `maxSize` – maximální výška oblasti při změně velikosti
      · `size` – aktuální výška oblasti
      · `closing` – udává, zda lze oblast uzavřít
      · `closed` – tuto vlastnost má, pouze pokud byla oblast uzavřena. V tomto případě má vlastnost vždy hodnotu `true` a oblast již není obsažena ve stránce (je skryta, nebo zcela odstraněna).
    * `south` – objekt s informacemi o dolní oblasti, má stejnou strukturu jako objekt s informacemi o horní oblasti (`north`)
    * `east` – objekt s informacemi o pravé oblasti, má stejnou strukturu jako objekt s informacemi o horní oblasti, ale vlastnosti `defaultSize`, `minSize`, `maxSize` a `size` udávají šířky
    * `west` – objekt s informacemi o levé oblasti, má stejnou strukturu, jako objekt s informacemi o pravé oblasti

    * `denyClosingComponentsWithDisabledClosing` – pravdivostní hodnota určující zda lze zavírat oblast s komponentem bez zavírací ikonky a společně s ní i daný komponent.

- `getComponentPositions()`
  - Popis: Metoda pro získání reference na objekt s informacemi o pozicích komponentů na stránce. Referencovaný objekt není dovoleno měnit.
  - Vrací referenci na objekt s informacemi o pozicích komponentů na stránce, který má následující veřejné vlastnosti:
    * `north` – pole sloupců v horní oblasti. Každá položka tohoto pole obsahuje pole řádků. Každá položka pole řádků obsahuje objekt s vlastností id, což je id komponentu na daném řádku.
    * `east` – pole sloupců v pravé oblasti, struktura jako u north.
    * `south` – pole sloupců v dolní oblasti, struktura jako u north
    * `west` – pole sloupců v levé oblasti, struktura jako u north
    * `center` – pole sloupců v centrální oblasti, struktura jako u north
    * `byId` – pole indexované id komponentů. Každá položka tohoto pole obsahuje objekt s následujícími vlastnostmi:
      · `area` – umístění oblasti, ve které se komponent nachází
      · `col` – sloupec, ve kterém se komponent nachází
      · `row` – řádek, ve kterém se komponent nachází

- `getComponentPrefsPresence()`
  - Popis: Metoda pro získání reference na pole s informacemi o tom, které komponenty mají uživatelská nastavení (`UserPref`). Pole je určeno pouze pro čtení a není dovoleno jej měnit.
  - Vrací referenci na pole indexované pomocí id komponentů. Každá položka tohoto pole je objekt s následujícími vlastnostmi:
    * `havePrefs` – informace o tom, zda má komponent nějaká nastavení
    * `visiblePrefs` – informace o tom, zda má komponent nastavení, která jsou zobrazena ve formuláři (zda má formulař s nastavením)
    * `hiddenPrefs` – informace o tom, zda má komponent nějaká nastavení, která nejsou zobrazena ve formuláři (`dataType="hidden"`)

- `getComponentPrefs()`
  - Popis: Metoda pro získání reference na pole s uživatelskými nastaveními komponentů. Získanou referenci nelze využít ke změně nastavení komponentů.
  - Vrací referenci na pole indexované pomocí id komponentů. Každá položka tohoto pole je pole uživatelských nastavení daného komponentu indexované názvy proměnných uživatelského nastavení (atribut `name` elementu `UserPref`). Každá položka pole uživatelských nastavení komponentu je objekt s následujícími veřejnými vlastnostmi:
    * `dataType` – datový typ položky (`'string'`, `'bool'`, `'enum'`, `'list'` nebo `'hidden'`)
    * `displayName` – zobrazované jméno položky nastavení
    * `required` – informace o tom, zda je položka vyžadována

    * `value` – hodnota položky
    * `enumValues` – pouze u typu enum, obsahuje položky výběru v poli objektů s následujícími vlastnostmi:
        · `displayValue` – zobrazovaná položka výběru
        · `value` – hodnota položky výběru

- `getComponentLocalizedText(componentId, textName)`
  - Popis: Metoda pro získání lokalizovaného textu komponentu
  - Parametry:
    * `componentId` – id komponentu, jehož lokalizovaný text se má získat
    * `textName` – Název lokalizovaného textu, který se má získat
  - Vrací řetězec s požadovaným lokalizovaným textem

- `getComponentLocales(componentId)`
  - Popis: Metoda pro získání informací o lokalizaci (locales) pro komponent
  - Parametry:
    * `componentId` – id komponentu, jehož lokalizační informace se mají získat
  - Vrací lokalizační informace pro komponent v objektu s následujícími vlastnostmi:
    * `lang` – jazyk
    * `country` – země

- `registerEventListener(componentId, eventName, eventHandler)`
  - Popis: Metoda pro registraci posluchače událostí (listeneru) do observeru. Zaregistruje funkci pro reakci na událost pomocí metody `observer.addListener()` a zajistí, že při uzavření komponentu bude tato funkce odregistrována.
  - Parametry:
    * `componentId` – id komponentu, který posluchače registruje
    * `eventName` – název události
    * `eventHandler` – funkce, která bude zaregistrována jako posluchač událostí (tato funkce bude volána při dané události)

- `unregisterEventListener(componentId, eventName, eventHandler)`
  - Popis: Metoda pro odregistraci posluchače událostí z observeru. Odregistruje funkci pro reakci na událost pomocí metody `observer.removeListener()`
  - Parametry:
    * `componentId` – id komponentu, který posluchače odregistruje
    * `eventName` – název události
    * `eventHandler` – funkce, která bude odregistrována

- `showError(title, text, opt_arg1, opt_arg2, ...)`
  - Popis: Metoda umožňuje zobrazit modální dialog s chybovou zprávou.
  - Parametry:
    * `title` – titulek dialogového okna
    * `text` – text chybové zprávy; může obsahovat zástupné proměnné ve tvaru `$X`, kde `X` je pořadí volitelného argumentu (`opt_argX`, číslováno od 1); tyto proměnné jsou nahrazeny za skutečné hodnoty argumentů
    * `opt_arg` – volitelné argumenty, jejichž hodnoty budou dosazeny do textu zprávy

Kromě skutečných metod má správce rozložení také „nahrazované“ metody, tedy metody, které v objektu neexistují, ale v komponentech je lze využívat. Tyto metody jsou při analýze XML s obsahem stránky textově nahrazeny za existující ekvivalenty (nelze je využít v dynamicky sestavovaném kódu). Jedná se o metody:

- `layoutManager.getMyComponent()`
  - nahrazeno za `layoutManager.getComponent('__MY_ID__')`
- `layoutManager.getMyComponentBody()`
  - nahrazeno za `layoutManager.getComponentBody('__MY_ID__')`
- `layoutManager.getMyComponentLocation()`
  - nahrazeno za `layoutManager.getComponentLocation('__MY_ID__')`

Zástupný symbol `__MY_ID__` je při analýze nahrazen řetězcovou hodnotou id komponentu. Pro využití je nutné jej umístit do uvozovek (příklad viz výše).

Zástupný symbol `__MODULE_ID__` je při analýze nahrazen u běžných komponent číselnou hodnotou id komponentu v uvozovkách, u gadgetů pouze číselnou hodnotou. Tento symbol je shodný se symbolem využívaným v Google gadgets.

Při analýze konfiguračního XML jsou nahrazeny také zástupné symboly za výchozí hodnoty proměnných uživatelských nastavení. Tyto zástupné symboly se skládají z řetězce `__UP_`, názvu proměnné a řetězce `__`. Např. element `<UserPref name="myName" default_value="jmeno"/>` definuje proměnnou uživatelského nastavení, za kterou lze využít zástupný symbol `__UP_myName__` a tento symbol bude nahrazen hodnotou „jmeno“. Tyto symboly jsou nahrazeny v titulku a obsahu komponentu.

Stejně jako zástupné symboly za proměnné jsou nahrazovány i zástupné symboly za lokalizovatelné texty. Tyto mají strukturu „`__MSG_`“ + název + „`__`“ a náhrady jsou definovány elementy `<msg>`. Zástupné symboly za lokalizované texty jsou nahrazovány nejenom v titulku a obsahu komponentu, ale také v atributu `display_name` položek uživatelského nastavení a v atributu `display_value` hodnoty výčtu v uživatelském nastavení.

Pokud k některému ze zástupných symbolů není nalezena příslušná proměnná ani lokalizovaný text, zástupný symbol je ponechán beze změny.

Správce rozložení také přidává do observeru další stavové objekty:

- `layoutAreasInfo` – obsahuje informace o oblastech na stránce. Komponenty jej mohou využít, ale nesmějí jej modifikovat. Referenci na něj lze získat výše uvedenou metodou, u jejíhož popisu jsou popsány i jednotlivé vlastnosti obsažených objektů.
- `layoutComponentPositions` – obsahuje informace o pozicích komponentů. Komponenty jej rovněž mohou využít, ale nesmějí modifikovat. Textový popis struktury a vlastností je uveden výše u popisu metody, která jej vrací. Při přesunu nebo uzavření komponentu správce rozložení automaticky upraví informace v tomto objektu a vyvolá příslušnou událost.
- `layoutComponentListeners` – je určen pro privátní použití správcem rozložení a obsahuje informace o funkcích, které mají komponenty registrované v observeru.
- `layoutComponentLocales` – obsahuje informace o lokalizačních informacích pro komponenty. Jedná se o pole indexované pomocí id komponentů, jehož každá položka je objekt s vlastnostmi `country` a `lang`.
- `layoutComponentMessagesPresence` – pole objektů indexované pomocí id komponentů obsahující informace o tom, které komponenty mají lokalizované texty. Každá položka tohoto pole je objekt s vlastností `haveMessages`.
- `layoutComponentMessages` – patří k objektu `layoutComponentMessagesPresence`, je rovněž privátní, a obsahuje lokalizované texty komponentů. Jedná se o pole indexované id komponentů, jehož každá položka je pole indexované názvy lokalizovaných textů. Každý text je uložen ve vlastnosti `content` příslušného objektu. Kromě této vlastnosti mají objekty také vlastnost `valid`, která je určena výhradně pro privátní využití správcem rozložení.
- `layoutComponentPrefsPresence` – obsahuje informace o tom, které komponenty mají uživatelská nastavení a o tom, zda mají formulář s nastavením a skrytá nastavení. Tento objekt, ani objekt se samotnými uživatelskými nastaveními, nesmějí být přímo modifikovány. Podrobný popis lze nalézt výše u metody pro získání reference (`getComponentPrefsPresence()`).
- `layoutComponentPrefs` – obsahuje uživatelská nastavení komponentů. Podrobný popis lze nalézt výše u metody pro získání reference (`getComponentPrefs()`).

Při manipulaci se všemi poli je třeba obezřetnost, protože Ext JS modifikuje konstruktor `Array()` a každé pole má metodu `remove()`. Při procházení cyklem „`for in`“ je třeba tuto metodu přeskočit.

Pro lokalizaci správce rozložení je využit obdobný princip, jako v knihovně Ext JS. Všechny vypisované lokalizovatelné textové řetězce jsou tedy ve veřejných vlastnostech třídy a pro lokalizaci se využívá změna prototypu. Aby se zde vlastnosti z prototypu nepředefinovaly, jsou definovány, pouze pokud první z nich není dostupná.

Je-li povolena podpora pro Google Gadgets, je nutné rozšířit vlastnosti správce rozložení (`KiWiLayoutManager`). Za tímto účel byla vytvořena funkce `KiWiLayoutManagerGadgetsPlugin`, jejímž vykonáním v kontextu existující instance správce rozložení (`KiWiLayoutManagerGadgetsPlugin.call(existingInstanceOfLayoutManager)`) dojde k zavedení nových metod a vlastností.

A.2 Události vyvolané správcem rozložení

Správce rozložení vyvolává různé události observeru, které mohou komponenty využít k detekci své pozice, změny rozměrů, pozice, nastavení spolupracujících komponentů apod. Následuje popis jednotlivých událostí, včetně popisu objektů, které jsou u nich předávány v parametrech:

- `layoutReady` – událost dokončení inicializace stránky, lze využít jako náhradu za metodu `Ext.onReady()`
- `layoutChanged` – událost změny rozložení stránky. Jako parametr je předán objekt, který má vždy vlastnost `action` (řetězec), udávající typ změny, a další vlastnosti s upřesňujícími informacemi dle typu změny. Následuje výpis možných hodnot vlastnosti `action` s popisy a dalších vlastností příslušejících k dané hodnotě:
  - `areaWillBeCollapsed` – tělo oblasti bude skryto (`beforecollapse`)
    - `id` – id oblasti (např. „`northArea`“)
    - `areaLocation` – umístění oblasti (např. „`north`“)
  - `areaCollapsed` – tělo oblasti skryto
    - `id` – id oblasti (např. „`northArea`“)
    - `areaLocation` – umístění oblasti (např. „`north`“)
  - `areaWillBeExpanded` – skryté tělo oblasti bude zobrazeno (`beforeexpand`)
    - `id` – id oblasti (např. „`northArea`“)
    - `areaLocation` – umístění oblasti (např. „`north`“)
  - `areaExpanded` – skryté tělo oblasti zobrazeno
    - `id` – id oblasti (např. „`northArea`“)
    - `areaLocation` – umístění oblasti (např. „`north`“)
  - `areaRemoved` – oblast odstraněna (zavřena)
    - `areaId` – id oblasti (např. „`northArea`“)
    - `areaLocation` – umístění oblasti (např. „`north`“)
  - `areaResized` – velikost oblasti změněna
    - `id -- id oblasti` (např. „`northArea`“)
    - `areaLocation` – umístění oblasti (např. „`north`“)
    - `adjWidth` – skutečná nová šířka (po přizpůsobení boxu)
    - `adjHeight` – skutečná nová výška (po přizpůsobení boxu)
    - `rawWidth` – nastavená šířka (viz `Ext.panel` resize event)
    - `rawHeight` – nastavená výška
  - `componentMoved` – komponent přesunut
    - `id` – id komponentu
    - `oldArea` – oblast, odkud byl komponent přesunut
    - `oldCol` – sloupec, odkud byl komponent přesunut
    - `oldRow` – řádek, odkud byl komponent přesunut
    - `newArea` – oblast, kam byl komponent přesunut
    - `newCol` – sloupec, kam byl komponent přesunut
    - `newRow` – řádek, kam byl komponent přesunut

  - `componentRemoved` – komponent odstraněn (zavřen)
    - `id` – id komponentu
    - `oldArea` – oblast, odkud byl komponent odstraněn
    - `oldCol` – sloupec
    - `oldRow` – řádek
  - `componentWillBeMaximized` – komponent bude maximalizován (zobrazen přes celou pracovní plochu)
    - `id` – id komponentu
    - `areaLocation` – oblast, ve které se komponent nachází
  - `componentWillBeMinimized` – komponent bude minimalizován (obnovena jeho původní velikost)
    - `id` – id komponentu
    - `areaLocation` – oblast, ve které se komponent nachází
  - `componentMaximized` – komponent byl maximalizován
    - `id` – id komponentu
    - `areaLocation` – oblast, ve které se komponent nachází
  - `componentMinimized` – komponent byl minimalizován
    - `id` – id komponentu
    - `areaLocation` – oblast, ve které se komponent nachází

- `layoutComponentPreferencesChanged` – událost změny hodnot uživatelských nastavení komponentu. Jako parametr je předán objekt s následujícími vlastnostmi:
  - `id` – id komponentu
  - `oldValues` – původní hodnoty v poli objektů indexovaném názvy proměnných uživatelského nastavení. Každý objekt v tomto poli má vlastnosti:
    - `dataType` – typ pole
    - `value` – hodnota
  - `newValues` – reference na pole objektů s novými hodnotami indexované názvy proměnných uživatelského nastavení. Každý objekt v tomto poli má vlastnosti:
    - `dataType` – typ pole (`'string'`, `'bool'`, `'enum'`, `'list'` nebo `'hidden'`)
    - `value` – hodnota
    - `displayName` – zobrazovaný název pole
    - `required` – informace o tom, zda se jedná o povinné pole
    - `enumValues` – pouze u typu `enum`, obsahuje položky výběru v poli objektů s následujícími vlastnostmi:
      - `displayValue` – zobrazovaná položka výběru
      - `value` – hodnota položky výběru
  - `changedByFormSaving` – vlastnost, která určuje, zda bylo nastavení změněno uložením příslušného formuláře (`true`), nebo programově skriptem (`false`)
  - `changedPrefName` – pouze pokud bylo nastavení změněno skriptem, obsahuje název změněné položky nastavení

Událost přesunu komponentu je vyvolána se zpožděním 200ms, čímž se obchází nedokonalost knihovny Ext JS, která způsobuje problémy při přesunu komponentu do prázdného

sloupce (nedojde k přizpůsobení šířky komponentu). Po 100ms dojde k fixaci chyby v rozložení stránky a po 200ms k vyvolání události.

Příklad využití událostí od správce rozložení:

```
<script type="text/javascript">
  observer.addListener("layoutReady", function(event) {
    layoutManager.getComponentBody("__MY_ID__").dom.innerHTML
      += "Stranka inicializovana! ";
  })
  layoutManager.registerEventListener("__MY_ID__",
      "layoutChanged", function(event) {
    if (event.action == "componentMoved") {
      if (event.id == "__MY_ID__") {
        layoutManager.getComponentBody("__MY_ID__").dom.innerHTML
          += "Moje nove umisteni je "
          + layoutManager.getMyLocation().area
          + " oblast " + layoutManager.getMyLocation().col
          + " sloupec " + layoutManager.getMyLocation().row
          + " radek. ";
      }
    }
  })
</script>
```

A.3 Struktura konfiguračního XML s obsahem stránky

Pro množství volitelných elementů s výchozím chováním a sémantických omezení strukturu konfiguračního XML vysvětlím na příkladech. Začnu extrémními příklady s vysvětlením chování v případě nedefinovaných informací a následně vysvětlím podrobně jednotlivé části. Minimální obsah vypadá následovně:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pageData>
  <layoutSettings></layoutSettings>
</pageData>
```

V případě načtení tohoto XML bude zobrazena prázdná stránka s centrální oblastí, ve které bude 1 prázdný sloupec. <`pagedata`> je povinný kořenový element dokumentu. <`layoutSettings`> je povinný element, který obsahuje informace pro správce rozložení. Pokud některý z těchto elementů nebude obsažen, stránku nebude možné zobrazit.

Centrální oblast musí být vždy dostupná, pokud není v konfiguraci uvedena, bude vytvořena s 1 prázdným sloupcem.

Vložíme-li do stránky 2 prázdné komponenty, bude XML vypadat následovně:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pageData>
  <layoutSettings></layoutSettings>
  <component/>
  <component id="1"/>
</pageData>
```

Pokud komponent nemá uvedené umístění, bude umístěn na první volný řádek v nultém sloupci centrální oblasti. Jedná se o první volný řádek při současném stavu zpracování XML, je tedy možné, že bude následně nalezen komponent, který se má na řádku skutečně nacházet. Ten potom bude posunut na další volný řádek a stejně budou posunuty i další komponenty.

Pokud některý komponent nemá uvedenu oblast, nebo požadovaná oblast není dostupná, bude umístěn v centrální oblasti, pokud nemá uveden sloupec, bude v nultém sloupci dané oblasti. Pokud některý komponent ve sloupci nemá uveden řádek, skutečné umístění všech komponentů v daném sloupci lze považovat za nedefinované.

Pokud komponent nemá id, bude mu přiděleno vygenerované id. Proto by měl mít každý komponent své id uvedeno, protože vygenerované id by následně mohlo dělat problémy při ukládání informací na server (je jednoznačné pouze v rámci načtení stránky). Při nastavování id je třeba odlišovat, zda se jedná o běžný komponent nebo komponent typu Google Gadgets – komponenty typu Google Gadgets musí mít id celočíselné. Vygenerovaná id pro běžné komponenty začínají řetězcem „`layout`“, skript pro ukládání dat na server by je měl rozeznat a neukádat nastavení komponentů s těmito id. Vygenerovaná id pro Google Gadgets jsou celočíselná a jsou uložena v poli `gadgetsGeneratedIds`; tedy i jejich rozeznání je možné.

Nyní komponentům doplním atributy, které by všechny komponenty měly mít, a přiřadím jim i minimální obsah. Dále doplním definici oblasti se dvěma sloupci. Obsah XML bude nyní následující:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pageData>
<layoutSettings>
  <area location="center">
    <cols>
      <col width="50%"/>
      <col width="50%"/>
    </cols>
  </area>
</layoutSettings>
<component id="2" location="center" col="0" row="0">
  <content type="text/html">
   <![CDATA[
     Text v tele komponentu.
  ]]>
  </content>
</component>
<component id="1" location="center" col="1" row="0">
  <content type="text/html">
   <![CDATA[
     Text v tele komponentu.
  ]]>
  </content>
</component>
</pageData>
```

Pokud bude oblast definována bez sloupců, bude v ní vytvořen jeden sloupec. V oblasti je možné definovat libovolný počet sloupců, přičemž u každého je nutno definovat jeho šířku v procentech ze šířky oblasti. Pokud šířka nebude uvedena, sloupec nebude vytvořen a komponenty z něj budou přesunuty do nultého sloupce. Pokud bude součet šířek sloupců v oblasti menší než 100%, bude vedle sloupců prázdný prostor, pokud bude větší, zobrazení bude chybné.

V elementu `layoutSettings` může být také element, který způsobí, že nebude možné zavřít komponent bez zavíracího tlačítka. Pokud se uživatel pokusí zavřít oblast s takovýmto komponentem, při zavírání daného komponentu bude zobrazeno chybové hlášení a daný komponent ani oblast nebudou uzavřeny. Tento element může být prázdný, bez atributů, a jeho syntaxe je následující:

```
<denyClosingAreaWithComponentWithDisabledClosing/>
```

A.3.1 Podrobná definice oblasti

Maximálně podrobná definice oblasti je:

```
<area location="east" defaultSize="200" minSize="175" maxSize="400">
  <title>Pravy panel</title>
  <collapsing enabled="true" collapsed="false"/>
  <closing enabled="true"/>
  <cols>
    <col width="50%" padding="10"/>
    <col width="50%"/>
 </cols>
</area>
```

Vysvětlení atributů elementu area:

- `location`
  - umístění oblasti
  - povinný atribut, neuvedení vede k ignorování definice oblasti
  - možné hodnoty: `north`, `east`, `south`, `west` nebo `center`
- `defaultSize`
  - u severní a jižní oblasti výchozí výška v px
  - u východní a západní oblasti výchozí šířka v px
  - u centrální oblasti nemá význam
- `minSize`
  - u severní a jižní oblasti minimální výška v px
  - u východní a západní oblasti minimální šířka v px
  - u centrální oblasti nemá význam
- `maxSize`
  - u severní a jižní oblasti maximální výška v px
  - u východní a západní oblasti maximální šířka v px
  - u centrální oblasti nemá význam

`defaultSize`, `minSize` a `maxSize` slouží pro určení rozsahu změny šířky (výšky) oblasti tažením myší. Pokud jsou všechny tři hodnoty stejné, změna je zablokovaná (vůbec se nezobrazí zesílené okraje oblastí pro uchopení myší a změnu rozměrů). Výchozí hodnoty jsou:

- u severní a jižní oblasti:
  - `defaultSize = 100`
  - `minSize = 80`
  - `maxSize = 200`
- u východní a západní oblasti:
  - `defaultSize = 200`
  - `minSize = 175`
  - `maxSize = 400`

Element `title` slouží pro textový titulek oblasti. Pokud není uveden, je využita jedna mezera, pokud je uveden prázdný element <`title/`>, titulek je prázdný. Pokud je titulek prázdný a je zakázán collapsing i zavírání, dojde k zobrazení pouze úzkého záhlaví oblasti.

Element `collapsing` slouží pro povolení či zakázání skrývání těla oblasti pomocí ikonky v záhlaví oblasti. Atribut `enabled` udává, zda je skrývání povoleno, a pokud je povoleno, atribut `collapsed` udává, zda má být oblast po načtení stránky skryta. Možné hodnoty jsou „`true`“ (povoleno / skryta) a „`false`“ (zakázáno / zobrazena). Výchozí hodnoty pro okrajové oblasti jsou:

`enabled = "true"`
`collapsed = "false"`
U centrální oblasti je výchozí hodnota:
`enabled = "false"`

Element `closing` slouží pro povolení či zakázání zavření oblasti. Pokud má atribut enabled hodnotu „`true`“, oblast bude mít ikonku pro zavření. Při kliknutí na tuto ikonku bude zobrazen dotaz na potvrzení zavření oblasti a po případném potvrzení budou nejprve zavřeny všechny komponenty v oblasti (a vyvolány příslušné události) a následně bude uzavřena celá oblast. Centrální oblast nelze zavírat, tento element u ní bude ignorován. Výchozí hodnota je `enabled = "false"`.

Element `col` slouží k definici sloupce. Jeho atributy jsou:

- `width`
  - šířka sloupce v % (procenta oblasti zabraná sloupcem)
  - všechny sloupce v oblasti musejí mít dohromady 100%
- `padding`
  - okraje oblasti v px
  - výchozí hodnota je 10px

Element `cols` slouží pouze pro umístění elementů `col`, které byly vysvětleny výše.

A.3.2 Podrobná definice komponentu

Nejprve vysvětlím základní komponent, který je minimálně závislý na správci rozložení, a následně detailně vysvětlím komponent, včetně rozšířených možností umožňujících mimo jiné i kompatibilitu s Google Gadgets.

```
<component id="header2" location="north" col="1" row="0">
  <preferences>
    <title>Titulek v zahlavi komponentu</title>
    <collapsing enabled="true" collapsed="false"/>
    <closing enabled="true"/>
    <maximizing enabled="true"/>
    <fixedHeight value="100" scrolling="true"/>
  </preferences>
  <content type="text/html">
   <![CDATA[
     Textovy obsah komponentu
   ]]>
  </content>
</component>
```

Vysvětlení atributů elementu `component`:

- `id`
  - jednoznačné id komponentu (v rámci celého systému)
  - řetězcová hodnota

- `location`
  - umístění oblasti, ve které se má komponent nacházet
  - pokud daná oblast nebude dostupná, využije se centrální
  - možné hodnoty: `north`, `east`, `south`, `west` nebo `center`
- `col`
  - sloupec, ve kterém se má komponent nacházet
  - pokud nebude dostupný, využije se nultý
- `row`
  - řádek, ve kterém se má komponent nacházet
  - pokud nebude volný (obsadí jej jiný komponent se stejnou hodnotou, nebo dojde k posunu komponentem s nedefinovanou hodnotou), využije se první volný řádek

Element `preferences` slouží pro vizuální úpravu XML a přehlednost, jeho obsah může být beze změny významu uveden přímo v elementu `component`.

Element `title` slouží pro textový titulek komponentu. Pokud není uveden, je využita jedna mezera, pokud je uveden prázdný element <`title/`>, titulek je prázdný. Pokud je titulek prázdný a je zakázáno skrývání těla komponentu (collapsing), maximalizování (maximizing) i zavírání, dojde k zobrazení pouze úzkého záhlaví komponentu.

Element `collapsing` slouží pro povolení či zakázání skrývání těla komponentu pomocí ikony v jeho záhlaví. Atribut `enabled` udává, zda je skrývání povoleno, a pokud je povoleno, atribut `collapsed` udává, zda má být tělo komponentu po načtení stránky skryto. Možné hodnoty jsou „`true`“ (povoleno / skryto) a „`false`“ (zakázáno / zobrazeno). Výchozí hodnoty jsou:
`enabled = "true"`
`collapsed = "false"`

Element `closing` slouží pro povolení či zakázání zavření komponentu. Pokud má atribut `enabled` hodnotu `false`, komponent nebude mít ikonu pro zavření. Při kliknutí na tuto ikonu bude zobrazen dotaz na potvrzení zavření a po případném potvrzení bude komponent uzavřen a vyvolána příslušná událost. Pokud ikona není zobrazena, stále může dojít k zavření komponentu tak, že bude přesunut do zavíratelné oblasti a následně bude uzavřena celá oblast. Pro úplné zakázání zavření komponentu je nutno zakázat tento způsob zavření komponentu výše uvedeným způsobem. Výchozí hodnota je `enabled = "true"`.

Element `maximizing` slouží pro povolení či zakázání maximalizace komponentu (zobrazení přes celou pracovní plochu). Atribut `enabled` udává, zda je maximalizace povolena. Možné hodnoty jsou `"true"` pro povolení a `"false"` pro zakázání. Výchozí hodnotou je `"true"`.

Element `fixedHeight` slouží k nastavení pevné výšky komponentu. Jeho atribut `value` udává výšku komponentu v px a atribut `scrolling` udává, zda mají být v komponentu v případě potřeby zobrazeny posuvníky (`overflow="auto"`). Pokud má atribut `scrolling` hodnotu `false`, posuvníky se nezobrazí a obsah komponentu bude oříznut na danou oblast (`overflow="hidden"`). Tato vlastnost je v Ext JS nazvána `autoScroll`. Výchozí chování neomezuje výšku komponentu, která se přizpůsobuje aktuálním potřebám. Výchozí hodnota `scrolling` je „`true`“, aby se v případě potřeby zobrazil posuvník pro horizontální posun.

Element `content` obsahuje obsah těla komponentu. Atribut `type` je v této verzi správce rozložení zcela ignorován a předpokládá se „`text/html`“. Obsah komponentu musí být textový, a pokud se nejedná o prostý text bez speciálních znaků, musí být uzavřen v sekci CDATA. Obsah tohoto elementu je také analyzován a jsou v něm nahrazeny některé řetězce (viz výše).

Obsahuje-li komponent element `ModulePrefs`, je zpracováván jako gadget. V opačném případě je považován za běžnou komponentu.

V obsahu komponentu se nesmí vyskytovat HTML elementy s id začínajícím řetězcem „`ext`“ nebo „`layout`“, aby nedošlo ke kolizi s automaticky generovanými id a k následným chybám.

Elementy `script` obsažené v těle komponentu jsou v rámci zjednodušení analýzy omezeny tak, že smějí mít pouze atributy `type` a `src` (jiné atributy budou odstraněny).

Před 1. výskytem elementu <`script`> v obsahu komponentu by měl být nejméně jeden alfanumerický znak (písmeno nebo číslice anglické abecedy), a to buď jako text, nebo i jako součást nějaké značky či entity. Nebude-li tam, bude v IE na začátek obsahu komponentu automaticky přidána nezalomitelná mezera (` `). Toto opatření obchází chybu v IE, která způsobuje, že jsou pro prohlížeč skripty před 1. alfanumerickým znakem neviditelné.

A.3.3 Podrobná definice komponentu typu Google Gadget

Komponenty typu Google Gadgets mají místo elementu <`title`> uvedený element <`ModulePrefs`>, v jehož atributu `title` je možné titulek uvést:

```
<component id="9999" location="north" col="1" row="0">
  <preferences>
    <ModulePrefs title="Titulek v zahlavi komponentu">
      <Locale messages="http://example.com/ALL_ALL.xml"/>
      <Locale lang="en" country="us">
        <msg name="hello_world">Hello World.</msg>
      </Locale>
    </ModulePrefs>
    <collapsing enabled="true" collapsed="false"/>
    <closing enabled="true"/>
    <maximizing enabled="true"/>
  </preferences>
  <Content type="text/html" view="home">
   <![CDATA[
     Textovy obsah komponentu pro pohled HOME (vychozi)
   ]]>
  </Content>
  <Content type="text/html" view="canvas">
   <![CDATA[
     Textovy obsah komponentu pro pohled CANVAS
   ]]>
  </Content>
</component>
```

Význam atributů a elementů je totožný, jako bylo uvedeno v části A.3.2 – Podrobná definice komponentu. Připomeňme, že atribut `id` musí mít celočíselnou hodnotu. Mezi rozdíly patří:

- `ModulePrefs`
  - element může obsahovat další elementy (`Locale, Require, Optional, ...`), dle specifikace Google Gadgets[1]
  - atribut `title` určuje titulek komponentu
  - atribut `height` určuje výchozí výšku komponentu (může být za běhu změněna)
  - atribut `scrolling` není v současné době podporován a posuvník se zobrazí vždy, když je třeba
- `Locale`
  - element `ModulePrefs` může obsahovat více elementů `Locale`, přičemž každý z těchto elementů může mít uvedené atributy `lang` a `country` (oba jsou volitelné)
  - jazykové zprávy <`msg`> se mohou vyskytovat pouze uvnitř elementů `Locale`
- `Content` nebo `content`
  - element `Content` může mít uvedený atribut `view`, který určuje, pro který pohled je obsah platný (podporovány jsou pohledy `HOME` a `CANVAS`)
  - správce rozložení podporuje z důvodu kompatibility variantu s malým i velkým počátečním písmenem (*content* i *Content*)

Je-li uveden element `ModulePrefs`, může být v jeho atributech uveden titulek, výška a nastavení posuvníků komponentu. Ostatní atributy tohoto elementu budou ignorovány. Pokud není uveden element <`title`>, který má vyšší prioritu, využije se hodnota atributu `title` z elementu `ModulePrefs`. Pokud není uveden element <`fixedHeight`>, pro pevnou výšku se využije hodnota atributu `height` elementu `ModulePrefs`. Pokud není uveden element `fixedHeight` a je nastaven atribut `height` elementu `ModulePrefs`, využije se pro nastavení zobrazení posuvníků (viz atribut `scrolling` elementu `fixedHeight`) hodnota atributu `scrolling` elementu `ModulePrefs`. Není-li atribut `scrolling` uveden, využije se `scrolling="true"`.

A.3.4 Podrobná definice komponentu včetně rozšířených možností

Definice komponentu s rozšířenými možnostmi (pro větší smysluplnost uvedena bez elementu `fixedHeight`) je následující:

```
<component id="header2" location="north" col="1" row="0">
  <preferences>
    <title>Titulek s hodnotou promenne __UP_myVal__
      a lokalizovatelnou casti __MSG_myText__</title>
    <collapsing enabled="true" collapsed="false"/>
```

[1] Gadgets XML Reference – http://code.google.com/apis/gadgets/docs/reference.html nebo http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/Gadgets-API-Specification.html#gadgetsExtendedXSD

```
    <closing enabled="true"/>
    <maximizing enabled="true"/>
    <Locale lang="cs" country="CZ"/>
    <UserPref name="myName" display_name="Name" required="true"
      datatype="string" default_value="Jmeno"/>
    <UserPref name="myEnum" display_name="Colors"
      datatype="enum">
      <EnumValue value="Red" display_value="__MSG_red__ color"/>
      <EnumValue value="Green"/>
      <EnumValue value="Yellow" display_value="Yellow color"/>
    </UserPref>
    <UserPref name="myList" display_name="__MSG_colors__"
      datatype="list" default_value="Red|Green|Blue"/>
    <UserPref name="myVal">
    <messagebundle>
      <msg name="myText">lokalizovana cast</msg>
      <msg name="colors">Barvy</msg>
      <msg name="red">Cervena</msg>
    </messagebundle>
    <dragging enabled="false"/>
  </preferences>
  <content type="text/html" view="canvas">
   <![CDATA[
    Tento obsah bude preskocen, protoze je urcen pouze
    pro celou obrazovku a tento kod patri beznemu komponentu
    (kdyby se jednalo o Google Gadget, bude obsah uvazovan).
  ]]>
  </content>
  <content type="text/html" view="home">
   <![CDATA[
    Textovy obsah komponentu.
  ]]>
  </content>
</component>
```

Element `Locale` obsahuje informace o lokalizaci pro daný komponent. Atribut `lang` obsahuje kód jazyka dle ISO 639-1, atribut `country` obsahuje kód země dle ISO 3166-1 alpha-2. Pokud některý z atributů není uveden, využije se jeho výchozí hodnota (stejně jako v případě neuvedení celého elementu):

```
lang="en"
country="US"
```

Elementy `UserPref` slouží k definici proměnných pro uživatelské nastavení komponentu. Vysvětlení jednotlivých atributů těchto elementů je následující:

- `name`
  - název proměnné
  - může se skládat z písmen anglické abecedy, číslic a podtržítek (regulární výraz: ^[a-zA-Z0-9_]+$)
  - povinný atribut, není-li uveden, element bude ignorován
- `display_name`
  - zobrazovaný název proměnné

  - bude zobrazen jako popis příslušného pole ve formuláři
  - při neuvedení bude využit název proměnné

- `datatype`
  - datový typ proměnné
  - není-li uveden, využije se výchozí hodnota „`string`“
  - možné hodnoty:
    - `string`
      - řetězec
      - ve formuláři pole pro zadání textového řetězce (<`input type="text"`>)
    - `bool`
      - pravdivostní hodnota
      - ve formuláři zaškrtávací políčko
    - `enum`
      - hodnota výčtového typu
      - ve formuláři rozbalovací seznam (combobox), uživatel si vybírá jednu z možností
    - `list`
      - seznam uživatelem poskytovaných textových hodnot
      - ve formuláři dynamická tabulka s 1 sloupcem, ve které lze přidávat a mazat řádky. Prázdné řádky jsou mazány automaticky. Lze přesunovat řádky pomocí myši technikou „táhni a pusť“ (drag and drop). Lze řadit položky podle anglické abecedy.
      - dle potřeby se převádí mezi polem řetězců a řetězcem, ve kterém jsou jednotlivé hodnoty odděleny znakem „|“
    - `hidden`
      - řetězcová hodnota, která může být využita skriptem
      - nezobrazuje se ve formuláři
      - pokud jsou všechny `UserPref` tohoto typu, negeneruje se formulář

- `required`
  - udává, zda je hodnota vyžadována
  - pokud je hodnota vyžadována, nelze uložit obsah formuláře s nastavením bez uvedení hodnoty

- `default_value`
  - výchozí hodnota proměnné
  - pokud není uveden, využije se prázdný řetězec (u datového typu `bool` hodnota `false`)

Z jednotlivých proměnných definovaných elementy `UserPref` (kromě těch, které mají datový typ `hidden`) je sestaven formulář s uživatelskými nastaveními komponentu. Po kliknutí na ikonku pro nastavení (vedle zavíracího tlačítka) se tento formulář zobrazí uživateli nad tělem komponentu. Po nastavení hodnot může uživatel obsah formuláře uložit (přičemž

se formulář opět skryje), nebo formulář skrýt a uvést do stavu před jeho zobrazením. Při uložení formuláře je vyvolána příslušná událost.

Pokud je formulář zobrazen a dojde k programové změně hodnoty proměnné, změna se ihned zobrazí ve formuláři.

Struktura komponentu s vygenerovaným formulářem pro nastavení je odlišná od struktury komponentu bez tohoto formuláře. Komponent bez formuláře je tvořen objektem třídy `Ext.Panel` a tělo tohoto panelu obsahuje obsah komponentu. Pokud má komponent nastavení, jedná se opět o objekt třídy `Ext.Panel`, ale má nastaveno vnitřní rozložení (anchor layout) a jsou v něm obsaženy další 3 panely: panel s formulářem pro nastavení, oddělovací panel (obsahuje pouze oddělovač) a panel s obsahem komponentu, v jehož těle se obsah komponentu nachází. Pro přístup k obsahu komponentu je tedy vhodné používat metodu `getComponentBody()`, která vždy vrací tělo správného panelu.

Elementy `EnumValue` slouží k definici položek výběru v uživatelské proměnné datového typu `enum`. Jejich atributy jsou:

- `value`
  - hodnota výběru (jedna z možných hodnot proměnné)
  - povinný atribut, při neuvedení bude element ignorován
- `display_value`
  - zobrazovaná hodnota výběru (zobrazí se uživateli v rozbalovacím seznamu)
  - nebude-li uvedena, využije se přímo hodnota výběru

Elementy `msg` slouží k definici lokalizovaných textů pro komponent. Povinný atribut `name` obsahuje název lokalizovaného textu (neuvedení způsobí ignorování elementu), pro jehož syntaxi platí stejná pravidla jako pro stejnojmenný atribut `UserPref`. Textový obsah elementu obsahuje lokalizovaný text. Tyto texty lze využít v titulku a obsahu komponentu, zobrazovaném názvu uživatelské proměnné a zobrazované položce výběru v uživatelské proměnné typu `enum`. Lze je také využít ve skriptech v komponentu, kde je možné získat jejich obsah pomocí metody `getComponentLocalizedText()`.

V případě Google Gadgets je nutné elementy `msg` vnořit do elementu `Locale` v části `ModulePrefs`. U běžných komponentů mohou být elementy `msg` umístěny přímo v elementu `component` nebo mohou být vnořeny do dalších elementů, např. do elementu `messagebundle` nebo `preferences`.

Element `dragging` slouží k povolení či zakázání uchopení komponentu a tažení. Pokud má jeho atribut `enabled` hodnotu „`true`“, vynutí zobrazení záhlaví komponentu. Pokud má hodnotu „`false`“, komponent nebude možno uchopit a přesunout.

U elementu `content` je analyzován také atribut `view`. Je-li podpora gadgets vypnuta a má-li tento atribut hodnotu „`canvas`“ nebo „`preview`“ a v komponentu se vyskytuje ještě další element `content`, daný element je ignorován a analyzuje se další element `content`.

A.4 Rekapitulace základů rozhraní pro JavaScript v komponentech

V této podkapitole uvedu základy rozhraní pro JavaScript v komponentech vysvětlené na příkladech. Tyto příklady lze využít pro rychlé zorientování v relativně složitém API.

U běžných komponentů jsou k dispozici dva globální objekty. U komponentů typu Google Gadgets pak existuje ještě objekt třetí, který implementuje Gadgets API:

- layoutManager
- observer
- gadgets (pouze u Google Gadgets)

Komponenty typu Google Gadgets by měly využívat především Gadgets API[2], tj. objekt *gadgets*.

A.4.1 Metody pro běžné komponenty

Každý komponent je tvořen třídou, která dědí z `Ext.Panel`, a může tedy využívat všechny metody a vlastnosti této třídy kromě těch, které by mohly nevhodně ovlivnit rozložení stránky (např. `setWidth()`).

Svoje id získá běžný komponent pomocí zástupného symbolu `__MY_ID__`:
```
var myId = '__MY_ID__';
```

Přístup ke svému panelu běžný komponent získá následovně:
```
var myPanel = layoutManager.getComponent('__MY_ID__');
```
nebo ekvivalentní zkratkou (textově nahrazena, nelze využít v generovaném kódu):
```
var myPanel = layoutManager.getMyComponent();
```

K obsahu těla komponentu lze přistoupit:
```
layoutManager.getComponentBody('__MY_ID__').dom.innerHTML = 'Obsah';
```
nebo ekvivalentní zkratkou (textově nahrazena, nelze využít v generovaném kódu):
```
layoutManager.getMyComponentBody().dom.innerHTML = 'Obsah';
```

Svoji pozici komponent získá metodou:
```
var myPosition = layoutManager.getComponentPosition('__MY_ID__');
```
nebo ekvivalentní zkratkou (textově nahrazena, nelze využít v generovaném kódu):
```
var myPosition = layoutManager.getMyPosition();
```
a jednotlivé vlastnosti získaného objektu jsou:
```
var myArea = myPosition.area;
var myCol = myPosition.col;
var myRow = myPosition.row;
```

Referenci na svoje uživatelské nastavení komponent získá následovně:
```
var myPreferences = layoutManager.getComponentPreferences('__MY_ID__');
```

Položku svého uživatelského nastavení (nikoliv reference, ale kopie) získá např.:
```
var myName = getComponentPrefsItem('__MY_ID__', "myName");
```
a její datový typ a hodnotu následně získá z vlastností:
```
var myNameType = myName.dataType;
var myNameValue = myName.value;
```

[2] http://code.google.com/apis/gadgets/docs/reference/

Hodnotu položky uživatelského nastavení může změnit pomocí:
`layoutManager.setComponentPrefsItem('__MY_ID__', "myName", "Nové jméno");`

Obdobně jako sám k sobě může komponent přistoupit i k jiným komponentům na stránce na základě řetězcové hodnoty jejich id.

Kód, který má být proveden po načtení stránky, musí být uveden ve funkci, která je registrována jako reakce na událost „`layoutReady`“:

```
observer.addListener("layoutReady", function(event) {
// tělo funkce
})
```

Komponent může měnit hodnoty vlastností stavového objektu observeru:
`observer.update('__MY_ID__', 'nameOfUpdatedProperty', updatedValue);`
a může reagovat na událost jeho aktualizace:

```
layoutManager.registerEventListener('__MY_ID__',"stateChanged",function(ev)
{
// tělo funkce
})
```

Komponent může reagovat také na výše uvedené události správce rozložení a může do observeru přidávat i další události, přičemž je nutno dbát na to, aby nedošlo k interferenci komponentů. Více o přidání nové události viz `Ext.util.Observable`.

A.4.2 Metody pro komponenty typu Google Gadgets

Komponenty jsou do správce rozložení stránky vložené formou plovoucího rámu (`iframe`), proto je jejich chování mírně odlišné a doporučuje se využít metod *Gadgets API*[3], které s touto skutečností počítají.

Svoje id získá gadgets komponent pomocí zástupného symbolu `__MODULE_ID__`:
`var myId = '__MODULE_ID__';`

Pro uživatelská nastavení lze místo metod `layoutManager.getComponentPreferences` a `layoutManager.setComponentPrefsItem` využít modulu `gadgets.Prefs`[4].

Událost `layoutReady` lze zachytit pomocí `gadgets.util.registerOnLoadHandler`[5]:
`gadgets.util.registerOnLoadHandler(function() { /* tělo funkce */ })`

Funkce observeru lze využívat stejně, jako bylo uvedeno u běžných komponentů:
`observer.update('__MODULE_ID__', 'nameOfUpdatedProperty', updatedValue);`

Pro další metody odkazujeme na web *Gadgets API Reference*, který obsahuje i ukázky.

[3] http://code.google.com/apis/gadgets/docs/reference/
[4] http://code.google.com/apis/gadgets/docs/reference/#gadgets.Prefs
[5] http://code.google.com/apis/gadgets/docs/reference/#gadgets.util.registerOnLoadHandler

PŘÍLOHA B

Požadavky na server

Text této kapitoly je se svolením autora převzat z [27] a aktualizován tak, aby reflektoval provedené změny.

Správce rozložení, observer a skripty pro demonstraci jejich činnosti jsou uloženy v následujících souborech a adresářích:

- index.php
  - skript, který z požadavku prohlížeče a parametru v URL zjistí požadovaný jazyk a vygeneruje indexovou stránku webu.
  - předpokládá se, že bude nahrazen skriptem, který bude sloužit ke stejnému účelu, ale bude spolupracovat s jinými částmi serveru a poskytne tak širší funkcionalitu (např. změna jazyka při přihlášení uživatele)
- locale
  - adresář s lokalizačními skripty
  - V názvu každého lokalizačního skriptu je kód jazyka dle ISO 639-1 (zde nahrazen XX). Lokalizační skripty jsou následující:
    - layout-lang-XX.js
      - skript pro lokalizaci správce rozložení
    - layout-lang-XX.php
      - skript pro lokalizaci indexové stránky
  - Skripty pro angličtinu jsou výchozí a musejí být vždy dostupné. Minimální obsah adresáře je tedy následující:
    - layout-lang-en.js
    - layout-lang-en.php
- initialization.js
  - skript pro inicializaci stránky, který deklaruje globální proměnné a po inicializaci Ext JS skryje informace pro uživatele bez JavaScriptu a vytvoří objekty observeru a správce rozložení. Správci rozložení nastaví také zdrojovou adresu a zavolá jeho metodu pro načtení obsahu stránky.
  - v případě zapnuté podpory Google Gadgets provádí rozšíření vytvořeného objektu správce rozložení o nové vlastnosti a přiřazení konfigurace gadgets do vlastnosti `gadgetsConfig` správce rozložení
  - před využitím je nutná jeho úprava, při které musí být aktualizována adresa, ze které bude načítáno konfigurační XML
  - server jej může generovat i dynamicky, a optimalizovat tak práci s TinyMCE apod.
- KiWiGadgetsConfig.js
  - konfigurace prostředí Google gadgets

  - před použitím je nutná úprava cest u vlastnosti `rpcRelay`

- KiWiGadgetsLayoutManager.js
  - rozšíření správce rozložení stránky (KiWiLayoutManager) o podporu pro Google Gadgets

- KiWiGadgetsRPC.js
  - podpora RPC pro Google Gadgets; soubor je vkládán do plovoucího rámu (iframe) zanořeném v gadgetu; bližší informace přímo ve zdrojových kódech

- KiWiGadgetsURITemplates.js
  - knihovna pro modul `views` v Google Gadgets; provádí dosazování proměnných do URI šablon

- KiWiGadgetsWrapper.js
  - skript definující třídy `gadgets`, které slouží pro zajištění funkčnosti Google Gadgets
  - metody tříd využívají Ext JS a v částečné míře obalují metody správce rozložení
  - pro jeho využití musí mít správce rozložení zapnutou podporu Google Gadgets a být rozšířen o dané vlastnosti (nastaveno v souboru initialization.js); také je nutné načíst konfiguraci gadgets z KiWiGadgetsConfig.js

- KiWiLayoutManager.css
  - stylový předpis pro některé prvky generované správcem rozložení a pro maximalizaci komponentů

- KiWiLayoutManager.js
  - třída správce rozložení

- KiWiLayoutReadyTrigger.js
  - skript pro vyvolání události `layoutReady` (jeho separace od správce rozložení řeší problémy s některými prohlížeči)

- KiWiObserver.js
  - třída observeru

- PortalColumn.js
  - sloupec portálu, plně převzato z příkladů pro Ext JS (složka *examples*)
  - lze upravit index.php a využít verzi uloženou přímo v knihovně Ext JS (u nové verze Ext JS nutno zkontrolovat funkcionalitu).

- portal.css
  - stylový předpis portálu, plně převzato z příkladů pro Ext JS (složka *examples*)
  - lze upravit index.php a využít verzi uloženou přímo v knihovně Ext JS

- Portal.js
  - portál – panel se sloupci umožňující přesuny komponentů. Je využit jako součást správce rozložení. Plně převzato z příkladů pro Ext JS (složka *examples*)
  - lze upravit index.php a využít verzi uloženou přímo v knihovně Ext JS (u nové verze Ext JS nutno zkontrolovat funkcionalitu)
- Portlet.js
  - panel komponentu, plně převzato z příkladů pro Ext JS (složka *examples*)
  - lze upravit index.php a využít verzi uloženou přímo v knihovně Ext JS (u nové verze Ext JS nutno zkontrolovat funkcionalitu)
- progress-bar.css
  - stylový předpis pro progressbar, plně převzato z příkladů pro Ext JS (složka *examples*)
- test.php
  - skript, který zasílá testovací konfigurační XML správci rozložení
  - určen pro demonstraci činnosti
  - jeho funkcionalitu nahradí jiná část serveru

Na serveru musí být umístěny výše uvedené soubory a adresáře (kromě .php souborů, které pravděpodobně budou nahrazeny) a následující adresáře:

- ext-latest
  - knihovna Ext JS
  - nutno instalovat včetně examples (využity sdílené obrázky, lze umístit jinam a upravit KiWiLayoutManager.css)
  - lze získat z http://extjs.com/products/extjs/download.php
  - nutno rozšířit o Ext.ux.TinyMCE. Toto rozšíření lze stáhnout z http://extjs.com/forum/showthread.php?t=24787 nebo z http://blogs.byte-force.com/xor/tinymce/index.html
- tinymce
  - editor TinyMCE
  - lze získat z http://tinymce.moxiecode.com/download.php, stejně jako jazykové soubory
  - rovněž je možné správce rozložení provozovat bez podpory TinyMCE; v tomto případě stačí odebrat příslušné řádky hlavičky indexové stránky

Server musí generovat konfigurační XML a indexovou stránku, která musí být v kódování UTF-8 a v hlavičce musí mít nejméně níže uvedený obsah. Pořadí řádků v uvedeném obsahu indexové stránky je nutné zachovat. Ke generování indexové stránky lze využít soubor index.php, který je součástí projektu, nebo k tomuto účelu vytvořit vlastní skript.

B.1 Příklad vygenerované indexové stránky s detailním popisem

```
1 <head>
2   <title>Titulek stranky</title>
3   <meta http-equiv="content-language" content="cs" />
4   <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
5   <link rel="stylesheet" type="text/css"
        href="./ext-latest/resources/css/ext-all.css" />
6   <!-- GC -->
7     <!-- LIBS -->
8     <script type="text/javascript"
          src="./ext-latest/adapter/ext/ext-base.js"></script>
9   <!-- ENDLIBS -->
10  <script type="text/javascript" src="./ext-latest/ext-all.js"></script>
11  <script type="text/javascript"
        src="./ext-latest/ux/miframe-min.js"></script>
12  <script type="text/javascript"
        src="./tinymce/jscripts/tiny_mce/tiny_mce.js"></script>
13  <script type="text/javascript"
        src="./ext-latest/ux/Ext.ux.TinyMCE.min.js"></script>
14  <script type="text/javascript"
        src="./ext-latest/build/locale/ext-lang-cs-min.js"></script>
15  <script type="text/javascript" src="Portal.js"></script>
16  <script type="text/javascript" src="PortalColumn.js"></script>
17  <script type="text/javascript" src="Portlet.js"></script>
18
19  <link rel="stylesheet" type="text/css" href="portal.css" />
20  <link rel="stylesheet" type="text/css" href="KiWiLayoutManager.css" />
21  <link rel="stylesheet" type="text/css" href="progress-bar.css" />
22
23  <script type="text/javascript" src="KiWiObserver.js"></script>
24  <script type="text/javascript" src="KiWiLayoutManager.js"></script>
25  <script type="text/javascript" src="KiWiGadgetsLayoutManager.js"></script>
26  <script type="text/javascript" src="./locale/layout-lang-cs.js"></script>
27  <script type="text/javascript" src="KiWiGadgetsConfig.js"></script>
28  <script type="text/javascript" src="initialization.js"></script>
29 </head>
30 <body>
31   <div id="layoutForUsersWihnoutJS">
32     Text pro uzivatele bez podpory JavaScriptu.
33   </div>
34   <div id="layoutLoading">
35     <h1>Nadpis stranky</h1><br/>
36     Stranka se nacita, cekejte prosim ...<br/><br/>
37     <div id="layoutLoadingProgress"></div>
38   </div>
39 </body>
```

Na řádku 5 je ke stránce připojen stylový předpis knihovny Ext JS a za ním následují jednotlivé skripty této knihovny. Na řádku 11 je rozšíření potřebné pro Ext.ux.TinyMCE, jehož skript se nachází na řádku 13. Jádro TinyMCE je vloženo na řádku 12.

Na řádku 14 je lokalizační skript pro knihovnu Ext JS. V uvedeném příkladu je česká lokalizace. Pokud by tento skript nebyl uveden, využije se výchozí jazyk, kterým je u Ext JS angličtina.

Na řádcích 15 – 17 jsou součásti rozšíření Ext.ux.Portal převzaté z příkladu Portal v knihovně Ext JS. Toto rozšíření je využíváno správcem rozložení.

Na řádku 19 je připojen stylový předpis správce rozložení, za ním následuje stylový předpis pro stránku a stylový předpis pro indikátor průběhu načítání (progressbar). Po těchto stylových předpisech následuje skript s observerem a skript se správcem rozložení.

Na řádku 26 je lokalizační skript pro správce rozložení. V uvedeném příkladu je česká lokalizace, pokud by tento skript nebyl uveden, využijí se výchozí anglické texty.

Na řádku 27 je umístěna konfigurace pro Google Gadgets a za ní následuje skript pro inicializaci stránky.

Tělo stránky musí vždy obsahovat nejméně 2 - 3 elementy (dle nastavení), jejichž význam je následující:

- `<div id="layoutForUsersWihnoutJS"/>`
  - element s informacemi pro uživatele bez podpory JavaScriptu
  - bude automaticky skryt ihned po inicializaci Ext JS
- `<div id="layoutLoading"/>`
  - element s informacemi zobrazenými v průběhu načítání stránky
  - bude automaticky skryt před koncem sestavování stránky (ve chvíli, kdy se bude nacházet v pozadí)
- `<div id="layoutLoadingProgress"/>`
  - element, ve kterém bude při načítání stránky zobrazen indikátor průběhu načítání (progressbar)
  - musí být umístěn v elementu `<div id="layoutLoading"/>`
  - musí být prázdný
  - pokud je zobrazení indikátoru průběhu načítání deaktivováno nastavením vlastnosti `showLoadingProgress` na `false`, nemusí být uveden

Veškeré další elementy přidané do těla stránky se musejí nacházet ve výše uvedených elementech. Mimo tyto elementy nesmí být v těle stránky žádný jiný obsah.